Executability of scenarios in Petri nets

Robert Lorenz [a,*], Gabriel Juhás [b], Robin Bergenthum [a], Jörg Desel [a], Sebastian Mauser [a]

[a] *Lehrstuhl für Angewandte Informatik, Katholische Universität, Eichstätt-Ingolstadt, 85071 Eichstätt, Germany*

[b] *Faculty of Electrical Engineering and Information Technology, Slovak University of Technology, Bratislava, Slovakia*

Keywords:
Place/transition Petri net
Inhibitor net
Partial order
Stratified order structure
Partial order semantics
Causal semantics

A B S T R A C T

In this paper, we show that it can be tested in polynomial time as to whether a scenario is an execution of a Petri net. This holds for a wide variety of Petri net classes, ranging from elementary nets to general inhibitor nets. Scenarios are given by causal structures expressing causal dependencies and concurrency among events. In the case of elementary nets and of place/transition nets, such causal structures are partial orders among transition occurrences. For several extended Petri net classes, the extension of partial orders to stratified order structures is considered.

The algorithms are based on the representation of the non-sequential behavior of Petri nets by so-called *token flow functions* and a characterization of Petri net executions called *token flow property*. This property allows nontrivial transformations into flow optimization problems, which can be solved in polynomial time. The paper is a revised, consolidated and extended version of the conference papers [G. Juhás, R. Lorenz, J. Desel, Can I execute my scenario in your net?, in: G. Ciardo, P. Darondeau (Eds.), ICATPN, in: Lecture Notes in Computer Science, Springer, 2005, pp. 289–308; R. Lorenz, S. Mauser, R. Bergenthum, Testing the Executability of Scenarios in General Inhibitor Nets, in: ACSD, IEEE Computer Society, 2007, pp. 167–176] and includes parts of the habilitation thesis [R. Lorenz, Szenario-basierte Verifikation und Synthese von Perinetzen: Theorie und Anwendungen, Habilitation, 2006].

1. Introduction

Specifications of concurrent systems are often formulated in terms of scenarios expressing causal dependencies and concurrency among events. In other words, it is often part of the specification that some scenario should or should not be an execution of the system. Thus, it is natural to consider the following problem:

Input: A concurrent system model and a scenario.
Problem: Is the scenario an execution of the system model?

In this paper, we consider Petri net models of concurrent systems. Petri nets allow an explicit representation, and a distinction of concurrency and nondeterminism. They have a concise graphical representation and support a variety of formal analysis methods. Therefore, they are one of the best established formalisms for the study of concurrency and for the modeling of real distributed systems in many application areas, such as communication networks [4], web services [5], manufacturing systems [6] and business processes [7].

* Corresponding author. Tel.: +49 08421 931137.
E-mail addresses: robert.lorenz@ku-eichstaett.de (R. Lorenz), gabriel.juhas@stuba.sk (G. Juhás), robin.bergenthum@ku-eichstaett.de (R. Bergenthum), joerg.desel@ku-eichstaett.de (J. Desel), sebastian.mauser@ku-eichstaett.de (S. Mauser).

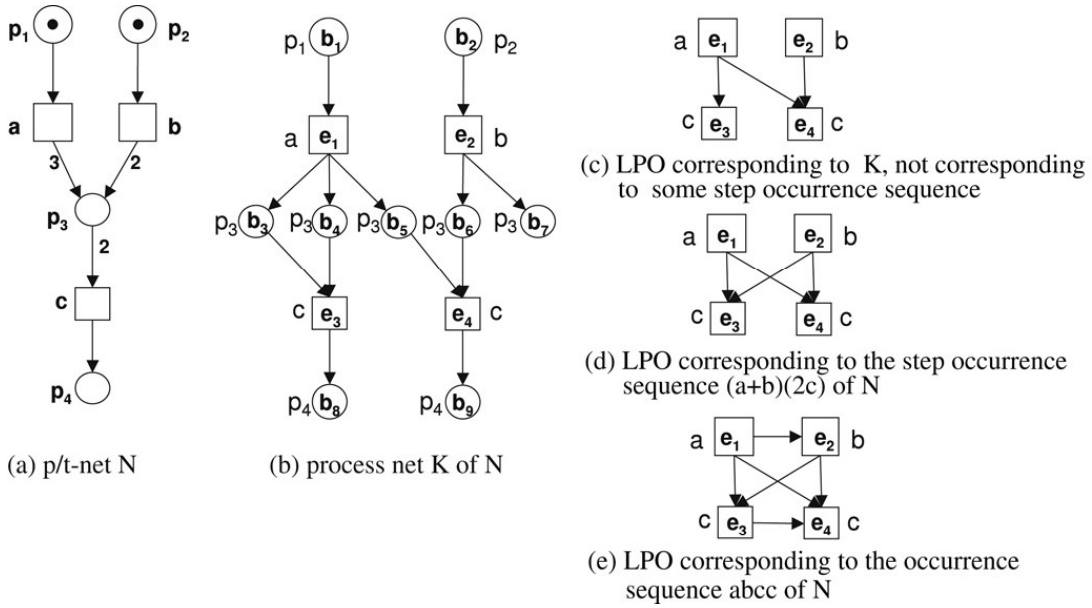


Fig. 1. A place/transition-net (p/t-net) together with executions w.r.t. different semantics. Each execution corresponds to a partial order of events labeled by transition names (representing transition occurrences), a so called *labeled partial order* (*LPOs*).

We consider the problem for several net classes. As it turns out, the solution is straightforward for elementary nets, but becomes complicated and non-trivial for place/transition Petri nets (p/t-nets) and their extensions.

An important variant of p/t-nets are Petri nets with inhibitor arcs. Petri nets with inhibitor arcs "are intuitively the most direct approach to increase the modeling power of Petri nets" [8] and have been found appropriate in various application areas [9]. In fact, it is well known that such nets are even equivalent to Turing-machines (w.r.t. their sequential behavior), and thus several decision problems, such as the reachability problem, which are decidable for p/t-nets, are undecidable for nets with inhibitor arcs. Therefore, it is an interesting and important question as to whether the considered problem can be (efficiently) solved for such nets.

Transforming the above question to Petri net models, we ask whether a given scenario is a possible execution of a given Petri net. There are different ways to represent executions of Petri nets, depending on the considered semantics. The most prominent concepts are *sequential semantics*, *step semantics*, *process semantics* and *causal semantics*. Sequential and step semantics are given by sets of occurrence sequences of single transitions resp. concurrent steps of transitions. They can be obtained by simply iterating the occurrence rule. Thus there is a straightforward test on executability of such sequences in linear time. The problem is that occurrence sequences of single transitions lack any information about independence and causality (Fig. 1(e)). Therefore, as soon as concurrency of events is specified, occurrence sequences of single transitions cannot be used for specification of scenarios. Occurrence sequences of concurrent steps of transitions allow us to specify causal dependency and concurrency of events only in a restricted way (Fig. 1(d)).

Process semantics are given by sets of process nets [10–14], which are Petri nets representing transition occurrences by events (transitions of process nets) with explicit pre-, post- and side-conditions (places of process nets). These conditions represent token occurrences (in the places of the original net) and other causal dependencies (for example context arcs) (Fig. 1(b)). Process nets can represent arbitrary concurrency relationships between events, and their defining properties can be verified in linear time. On the other hand, process nets are not very suitable for specification purposes, for two reasons. First, conditions are labeled by names of places of the model specified. Hence, it is not possible to specify that two events have to occur in some order, but it is rather necessary to state which place is responsible for establishing this order. So the specification already includes details of an implementation. The second disadvantage is that a process net determines the precise causality between events. Hence it is not possible to specify a scenario with two events that may either occur (causally) ordered or concurrently.

These problems can be overcome by considering causal semantics. Causal semantics are given by sets of appropriate causal structures expressing arbitrary concurrency relationships among events. In the case of p/t-nets, the causal structures are partial orders of events labeled by transition names (representing transition occurrences), so called *labeled partial orders* (*LPOs*) (Fig. 1(c)–(e)).[1] We interpret such a partial order between events as follows: If two events, e_1 and e_2, labeled by transitions t_1 and t_2, respectively, are ordered ($e_1 < e_2$) then t_1 may occur before t_2 or both may occur concurrently (concurrent occurrence includes sequential occurrence). If e_1 and e_2 are not ordered, then concurrent execution of t_1 and t_2 is demanded. That means, an LPO describes a possible observation of an execution where possibly not all concurrency is observed. Thus, quite a natural way to specify scenarios of a p/t-net is in terms of LPOs, which can (or cannot) be executions

[1] These LPOs are called *pomsets* (partially ordered multisets) in [15] and *partial words* in [16].

of the p/t-net. There are three equivalent characterizations of executions of p/t-nets, where only the third one leads to a polynomial test as to whether a given LPO is an execution:

(i) An LPO is *enabled* w.r.t. a p/t-net, if, for each cut of the LPO, the marking reached by firing all transitions corresponding to events smaller than the cut, enables the multi-set of transitions given by the cut (a cut is a maximal set of independent nodes). Unfortunately no efficient algorithm can immediately test LPOs to be enabled, because the number of cuts grows exponentially with the size of the LPO in general.

(ii) Process nets can be translated to LPOs by removing all conditions and keeping the partial order for the events (Fig. 1 (c)). We call such LPOs *runs*. An LPO is *executable* in a p/t-net, if it sequentializes (adds causality to) a run (Fig. 1(c)–(e)).[2] There is no efficient test as to whether an LPO is executable. This is because, with the number of choices, the number of runs also grows exponentially with the size of the p/t-net in general (the p/t-net belongs to the input of the considered problem).

(iii) In [1] we introduced the so called *token flow property* of LPOs. We showed that an LPO is enabled (resp. executable) if, and only if, it satisfies the token flow property w.r.t. a given p/t-net. We developed a polynomial algorithm to test LPOs to fulfill the token flow property, based on a transformation onto a flow maximization problem. The algorithm runs in time $O(q \cdot n \cdot g(n, e))$, where n and e are the number of nodes and edges of the LPO, q is the number of places of the p/t-net and $g(n, e)$ is the polynomial time bound of the flow maximization algorithm applied [19].
In [3] an even faster algorithm is presented, running in time $O(q \cdot g(n, e))$. But, in comparison to the first algorithm which exhibits a counter example in the negative case, this faster algorithm returns less information about the reasons for a negative answer originating from the structure of the p/t-net or of the LPO.

In the case of Petri nets with inhibitor arcs there are two different causal semantics leading to different causal structures representing executions. According to the so-called a posteriori semantics, executions are given by LPOs. They can be defined as enabled LPOs, analogously as in the p/t-net case. In the a priori semantics, as observed in [20,21], executions can be formally given as labeled stratified order structures (LSOs), a proper generalization of LPOs.[3] In [21] the most general notion of such nets, so-called PTI-nets, are considered. The authors develop process semantics for such nets, together with associated causal semantics given in terms of executable LSOs. As discussed in [14], for these process semantics and causal semantics, the important equivalence of executable and enabled LPOs does not carry over to LSOs and PTI-nets. That means, if one introduces the notion of *enabled LSOs* as a proper generalization of enabled LPOs in the obvious way, then there are LSOs which are enabled but not executable. Therefore, in [14] a modified definition of process semantics is proposed, leading to the equivalence of the notions of enabled and executable LSOs. The existence of such process semantics justifies the use of enabled LSOs as causal semantics of PTI-nets in this paper. Obviously, analogously to the case of LPOs, the notions of enabled and executable LSOs, again do not lead to efficient algorithms. In [2] we defined the *token flow property* of LSOs w.r.t. PTI-nets as a generalization of the respective notion for LPOs and p/t-nets, and show its equivalence to the notions of executions of enabled respectively executable LSOs. The polynomial algorithm is then again developed from the token flow property. It turns out that it can be based on an algorithm for the LPO case, and needs an additional check of inhibitor constraints. This additional check is performed through a transformation onto a flow *minimization* problem, which also allows efficient solution methods, running in time $g(n, e)$.

In Fig. 2, the relationship between the different characterizations of executions is depicted for p/t-nets (left part) and PTI-nets w.r.t. a priori semantics (right part).

In the conference paper [1] we presented a polynomial algorithm to answer the executability problem, when the system is given by a p/t-net. In the habilitation thesis [3], an alternative and faster algorithm is proposed, several possibilities to optimize both algorithm are discussed and applications are described. In the conference paper [2] these results are extended to p/t-nets with weighted inhibitor arcs (PTI-nets), the most general notion of Petri nets with inhibitor arcs, w.r.t. the a priori semantics. In this paper, we subsume these results in a consolidated and revised version. Moreover, we also adapt the theory for PTI-nets w.r.t. the a posteriori semantics, and give a brief overview on further net classes.

In the case of p/t-nets, the surprising message might not be the existence of polynomial algorithms, but the fact that this is not a trivial problem.

In fact, for elementary Petri nets or 1-safe p/t-nets, there exists an immediate algorithm to decide the problem, because a unique corresponding process net can be constructed from an LPO — if it exists. Given an LPO, we start by constructing the minimal conditions of the process given by the initial marking of the net. Then we iteratively choose a minimal event of the LPO, try to append it to the maximal conditions of the so-far constructed process, together with its post-conditions, and remove it from the LPO. Since, in elementary nets, a place can be marked by, at most, one token, there is always, at most, one possibility to append such an event. If it is not possible to append the event, or if token flow adds order to the LPO through appending the event, the LPO is no execution. The crucial point for p/t-nets is that, due to their non-safeness, there is always the choice between several tokens from the same place (in particular, there is not a unique process net corresponding to a given LPO, i.e. an LPO can sequentialize different runs).

---

2 It was shown in [17,18] that an LPO is enabled if, and only if, it is executable.

3 Stratified order structures were originally introduced independently in [22] (under the name prossets) and in [23] (under the name composets).

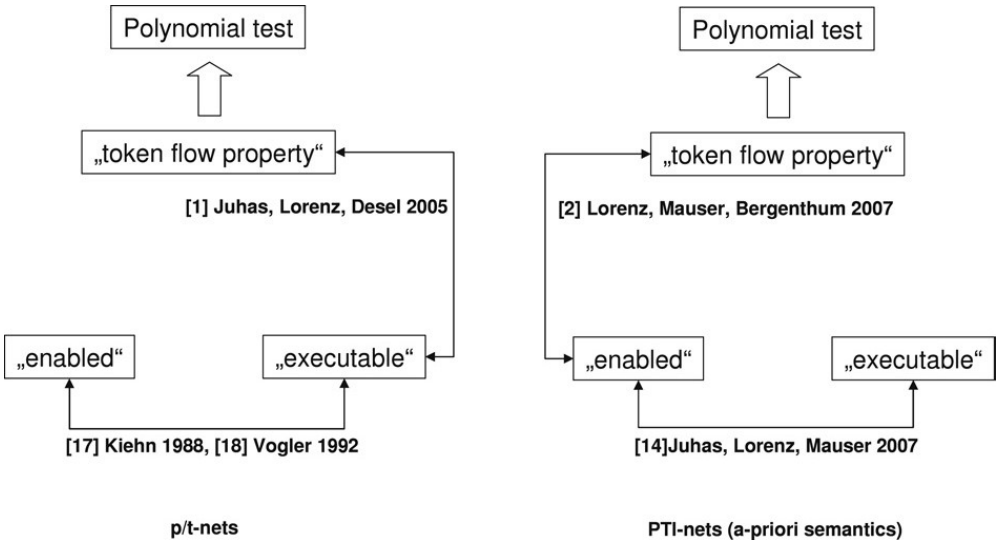


Fig. 2. Theorems in literature.

On the other hand, in the case of PTI-nets, the result is quite surprising, because for many Petri net problems the extension by inhibitor constraints complicates the solution by several degrees, or even leads to undecidability.

The structure of the remainder of this paper is as follows. In Section 2, we consider the executability problem for p/t-nets. We start with a brief discussion of causal semantics of p/t-nets (Section 2.1), then introduce the characterization of executions of p/t-nets called *token flow property* (Section 2.2) and present two polynomial algorithms to test the token flow property of a given LPO (Section 2.3). We also provide several heuristics to improve the time bounds of the algorithms (Section 2.4), compare the algorithms concerning efficiency and the possibility of fault analysis (Section 2.5) and briefly discuss related variants of the executability problem (Section 2.6). In Section 3, we discuss causal semantics of PTI-nets (Section 3.1) and generalize the theory to PTI-nets w.r.t. the a priori semantics (Section 3.2) and the a posteriori semantics (Section 3.3). That means we generalize the notions of LPOs enabled resp. fulfilling the token flow property w.r.t. p/t-nets, to LSOs (LPOs) enabled resp. fulfilling the token flow property w.r.t. PTI-nets, and present a polynomial algorithm to test the token flow property of a given LSO (LPO). Finally, in Section 4 we give an overview of the solution of the executability problem for the classes of elementary nets, elementary nets with (mixed) context (in the a posteriori and a priori semantics), p/t-nets with capacities (in the weak and strong semantics) and p/t-nets with unweighted inhibitor arcs (in the a posteriori and a priori semantics). Some conclusions and outlooks on future work are given in Section 5.

2. Place/transition-nets

In this section, we consider the problem of the executability of scenarios for place/transition-nets. We use $\mathbb{N}$ to denote the *nonnegative integers*. Given a finite set A, the symbol $|A|$ denotes the *cardinality* of A. A *multi-set* over A is a function $m : A \to \mathbb{N}$. For an element $a \in A$ the number $m(a)$ determines the number of occurrences of a in m. $\mathbb{N}^A$ is the set of all multi-sets over A.

A *directed graph* G is a tuple $G = (V, \to)$, where V is a finite set called its set of *nodes* and $\to \subseteq V \times V$ is a binary relation over V called its set of *arcs*. As usual, given a binary relation $\to$, we also write $a \to b$ instead of $(a, b) \in \to$. For $v \in V$ and $W \subseteq V$ we denote by ${}^\bullet v = \{v' \in V \mid v' \to v\}$ the *preset* of v, and by $v^\bullet = \{v' \in V \mid v \to v'\}$ the *postset* of v. ${}^\bullet W = \bigcup_{w \in W} {}^\bullet w$ is the *preset* of W and $W^\bullet = \bigcup_{w \in W} w^\bullet$ is the *postset* of W. A sequence of nodes $v_0 \dots v_n$ ($n \in \mathbb{N}$) with $v_{i-1} \to v_i$ for $i \in \{1, \dots, n\}$ is a *path* from v_0 to v_n. A path is *simple* if no node occurs twice. A path $v_0 \dots v_n$ with $v_0 = v_n$ is a *cycle*.

A *partial order* is a directed graph $(V, <)$, where $< \subseteq V \times V$ is an irreflexive, transitive binary relation. A *labeled partial order* (LPO) is a triple $(V, <, l)$, where $(V, <)$ is a partial order, and l is a *labeling function* on V (Fig. 1(c)–(e)). In this paper, a partial order is interpreted as an "earlier than"-relationship between events, which can be observed during an execution of a system.

Two different nodes (events) $v, v' \in V$ are called *independent* if $v \not< v'$ and $v' \not< v$. By $co_< \subseteq V \times V$ we denote the set of all pairs of independent nodes of V. A *co-set* is a subset $S \subseteq V$ fulfilling $\forall x, y \in S : x \, co_< \, y$. A *cut* is a maximal co-set. For a co-set S and a node $v \in V \setminus S$ we write $v < S$ ($v > S$), if $\exists s \in S : v < s$ ($\exists s \in S : v > s$), and $v \, co_< S$, if $\forall s \in S : v \, co_< s$. A node v is called *maximal* if $v^\bullet = \emptyset$, and *minimal* if ${}^\bullet v = \emptyset$.

A subset $W \subseteq V$ is called *closed* if $\forall v, v' \in V : (v \in W \wedge v' < v) \implies v' \in W$. For a closed subset $W \subseteq V$, the partial order $(W, <|_{W \times W})$ is called *prefix* of $(V, <)$, defined by W (as usual $R|_A$ denotes the *restriction* of a relation R onto a set A). The *closure* of a subset W is given by the set $W \cup \{v \in V \mid \exists w \in W : v < w\}$. The closure of a subset defines a prefix of a partial order. The node set of a prefix equals the closure of the set of its maximal nodes.

By $\lessdot \subseteq <$ we denote the smallest subset $<'$ of $<$ which fulfills $(<')^+ = <$ (as usual R^+ denotes the transitive closure of a relation R), called the *skeleton (or Hasse diagram) of* $<$.

Given two partial orders, $\mathrm{po}_1 = (V, <_1)$ and $\mathrm{po}_2 = (V, <_2)$, we say that po_2 *is a sequentialization of* po_1 if $<_1 \subseteq <_2$.

We use all notations defined for partial orders, also for LPOs. If $\mathrm{lpo} = (V, <, l)$ and $l : V \to X$, then for a subset $W \subseteq V$, we define the multi-set $l(W) \subseteq \mathbb{N}^X$ by $l(W)(x) = |\{v \in W \mid l(v) = x\}|$.

A *net* is a triple (P, T, F), where P is a finite set of *places*, T is a finite set of *transitions*, satisfying $P \cap T = \emptyset$, and $F \subseteq (P \cup T) \times (T \cup P)$ is a *flow relation*. The presets and postsets of (sets of) places and transitions are defined w.r.t. the directed graph $(P \cup T, F)$. For simplicity, we consider only nets in which every transition has a nonempty preset and postset.

A *place/transition-net* (shortly *p/t-net*) N is a quadruple (P, T, F, W), where (P, T, F) is a net, and $W : F \to \mathbb{N} \setminus \{0\}$ is a *weight function*. We extend the weight function W to pairs of net elements $(x, y) \in (P \times T) \cup (T \times P)$ satisfying $(x, y) \notin F$ by $W((x, y)) = 0$.

A *marking* of a p/t-net $N = (P, T, F, W)$ is a function $m : P \to \mathbb{N}$. A *marked p/t-net* is a pair (N, m_0), where N is a p/t-net, and m_0 is a marking of N, called *initial marking*. Fig. 1 (a) shows a marked p/t-net.

A multi-set (step) of transitions $\tau \in \mathbb{N}^T$ is *enabled to occur in a marking* m of N if $m(p) \geq \sum_{t \in T} \tau(t) W((p, t))$. If a step of transitions τ is enabled to occur in a marking m, then its *occurrence* leads to the new marking m' defined by $m'(p) = m(p) - \sum_{t \in T} \tau(t)(W((p, t)) - W((t, p)))$. We write $m \xrightarrow{\tau} m'$ to express that τ is enabled to occur in m and that its occurrence leads to m'.

A finite sequence of transition steps $\sigma = \tau_1 \ldots \tau_n$, $n \in \mathbb{N}$, is called *step occurrence sequence enabled in* m_0 *and leading to* m_n if there exists a sequence of markings $m_1, \ldots, m_n$ such that $m_0 \xrightarrow{\tau_1} m_1 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_n} m_n$. The marking m_n is said to be *reachable from the marking* m_0.

An *occurrence net* is a net $O = (B, E, G)$ such that $|{}^\bullet b|, |b^\bullet| \leqslant 1$ for every $b \in B$, and G^+ is a partial order on $B \cup E$. Places of an occurrence net are called *conditions* and transitions of an occurrence net are called *events*. The set of conditions which are minimal (maximal) according to G^+ is denoted by $\mathrm{Min}(O)$ ($\mathrm{Max}(O)$). Clearly, $\mathrm{Min}(O)$ and $\mathrm{Max}(O)$ are cuts w.r.t. G^+.

A *process* of (N, m_0) is a pair $K = (O, \rho)$, where O is an occurrence net and $\rho : B \cup E \to P \cup T$ is a labeling function with (i) $\rho(B) \subseteq P$ and $\rho(E) \subseteq T$, (ii) $\forall e \in E, \forall p \in P : |\{b \in {}^\bullet e \mid \rho(b) = p\}| = W((p, \rho(e)))$ and $\forall e \in E, \forall p \in P : |\{b \in e^\bullet \mid \rho(b) = p\}| = W((\rho(e), p))$ and (iii) $\forall p \in P : |\{b \in \mathrm{Min}(O) \mid \rho(b) = p\}| = m_0(p)$ (Fig. 1(b)).

2.1. Causal semantics of p/t-nets

In this subsection we briefly summarize known notions and results concerning the causal semantics of p/t-nets. As mentioned in the introduction, executions of p/t-nets are represented as enabled LPOs or executable LPOs.

The notion of executable LPOs is based on so called *runs* associated to a process net $K = (O, \rho)$ of a marked p/t-net (N, m_0). The *run* of (N, m_0) representing K is defined as the LPO $\mathrm{lpo}_K = (E, G^+|_{E \times E}, \rho|_E)$. A run is said to be *minimal* if it is not a sequentialization of another run.[4] An LPO $(V, \prec, l)$ is *executable* in (N, m_0) if there is a run $(V, <, l)$ of (N, m_0) with $< \subseteq \prec$, and *minimal executable* if it is a minimal run.

An LPO $\mathrm{lpo} = (V, \prec, l)$ is called *enabled (to occur) w.r.t.* (N, m_0) if, for every cut S of lpo and every $p \in P$:

$$m_0(p) + \sum_{v \in V \wedge v \prec S} (W((l(v), p)) - W((p, l(v)))) \geq \sum_{v \in S} W((p, l(v))).$$

Its occurrence leads to the marking $m'(p)$, given by

$$\begin{aligned} m'(p) &= m_0(p) + \sum_{v \in V} (W((l(v), p)) - W((p, l(v)))) \\ &= m_0(p) + \sum_{t \in T} l(V)(t)(W((t, p)) - W((p, t))). \end{aligned}$$

We write $m_0 \xrightarrow{\mathrm{lpo}} m'$ in this case. This definition can be equivalently formulated with cosets instead of cuts.

An equivalent characterization of enabled LPOs is through step occurrence sequences. A step sequence of transitions $\sigma = \tau_1 \ldots \tau_n$ can be identified with the LPO $\mathrm{lpo}_\sigma = (V, \prec, l)$, where $V = \bigcup_{i=1}^n V_i$ is a disjoint union and $l : V \to T$ with $l(V_i)(t) = \tau_i(t)$, and $\prec = \bigcup_{i<j} V_i \times V_j$. An LPO is enabled if, and only if, each step sequence sequentializing the LPO is a step occurrence sequence of (N, m_0). An enabled LPO is said to be *minimal enabled* if it is not the sequentialization of another enabled LPO.

It is clear by definition that if an LPO is enabled w.r.t. a marked p/t-net (N, m_0) and its occurrence leads to m', then every sequentialization of this LPO is enabled w.r.t. (N, m_0) and leads to m', too. Moreover, it can be easily shown that runs are enabled. This directly implies that executable LPOs are always enabled. The important result completing the relationship between enabled LPOs, runs and executable LPOs was proven in [17,18]. It states that if an LPO is enabled w.r.t. (N, m_0), then

[4] In an elementary net, having only arc weights and markings of value 0 and 1, every run is minimal.

it is also executable in (N, m_0). This implies, in particular, that the set of minimal runs of a marked p/t-net equals the set of its minimal enabled LPOs. Enabled resp. executable LPOs are also called *executions* in this paper, minimal enabled LPOs are also called *minimal executions*. Fig. 1 (c) shows a run of a p/t-net, which is not minimal. The LPOs shown in the parts (d) and (e) sequentialize this run.

2.2. Token flow property

In this subsection, we briefly restate the definitions and main results of the conference paper [1] concerning the characterization of Petri net executions by token flow functions. Since the focus of this paper is on algorithms, we omit the proofs here (they can be found in [1]).

From the last subsection, we have that an LPO is executable if, and only if, it is enabled. As argued in the introduction, these two notions of executions are not appropriate to deduce efficient algorithms for a test on executability. Therefore, we introduce the so called *token flow property* of LPOs w.r.t. a marked p/t-net (N, m_0). The token flow property is based on a new representation of the non-sequential behavior of p/t-nets by so-called *token flow functions*. In [1] we show that an LPO fulfills the token flow property w.r.t (N, m_0) if, and only if, it is executable in (N, m_0). In the next subsections we present polynomial tests of LPOs to check if they fulfill the token flow property. In the positive case, these tests compute a run of (N, m_0) sequentialized by this LPO.

Fix a marked p/t-net (N, m_0), $N = (P, T, F, W)$, and a place $p \in P$. Given an LPO $\text{lpo} = (V, \prec, l)$ with $l(V) = T$ we assign non-negative integers to its edges through a so-called *token flow function*. The aim is to find a token flow function χ, assigning values $\chi((v, v'))$ to edges (v, v') in such a way that there is a process with exactly $\chi((v, v'))$ post-conditions of v labeled by p which are also pre-conditions of v'. Thus, such a token flow function of lpo abstracts from the individuality of conditions of a process and encodes the flow relation of this process by natural numbers. That means, in particular, that $\chi((v, v'))$ equals the number of tokens which are first produced by the transition $l(v)$ and then consumed by the transition $l(v')$. It is possible to assign the value 0 to an edge. An LPO fulfills the *token flow property*, if there exists such a token flow function for every place p. In the positive case, the LPO sequentializes the run corresponding to the process encoded by the token flow functions.

In order to simplify the formal definition of the token flow property, we define an extension of $\text{lpo} = (V, \prec, l)$ by adding an initial node which is smaller than all nodes from V and is labeled by a new label. It represents a transition, producing the initial marking, and helps to avoid several case distinctions in the formal definitions.

Definition 1 (*Token Flow Function*). An LPO $\text{lpo}^0 = (V^0, \prec^0, l^0)$, where $V^0 = (V \cup \{v_0\})$, $v_0 \notin V$, $\prec^0 = \prec \cup (\{v_0\} \times V)$, and $l^0(v_0) \notin l(V)$, $l^0|_V = l$, is called 0-*extension of* $\text{lpo} = (V, \prec, l)$.

We define $\text{In}(v, \chi) = \sum_{v' \prec v} \chi((v', v))$ and $\text{Out}(v, \chi) = \sum_{v \prec v'} \chi((v, v'))$ for a function $\chi : \prec^0 \to \mathbb{N}$ and $v \in V^0$.

A function $\chi : \prec^0 \to \mathbb{N}$ is a *token flow function* of lpo, if it satisfies (**Tff**) $\forall v, v' \in V^0 : l(v) = l(v') \implies \text{In}(v, \chi) = \text{In}(v', \chi)$. $\text{In}(v, \chi)$ is the *intoken flow of* v *w.r.t.* χ and $\text{Out}(v, \chi)$ is the *outtoken flow of* v *w.r.t.* χ.

This definition differs from that in [1]. While in [1] token flow functions were defined, in general, as possible, we here additionally require property (**Tff**). This is more intuitive and does not restrict the setting or change the argumentations, since (**Tff**) is implicitly contained in the token flow property defined below. Each process $K = (O, \rho)$, $O = (B, V, G)$ of (N, m_0) defines so-called *canonical token flow functions* $\chi_p : \prec^0 \to \mathbb{N}$ of the run $(V, \prec, l)$, representing this process via $\chi_p((v, v')) = |\{b \in B \mid \rho(b) = p \wedge b \in v^\bullet \cap {}^\bullet v'\}|$ for each place p (denote $v_0^\bullet = \text{Min}(O)$) (Fig. 3). Canonical token flow functions obviously fulfill (**Tff**). By definition, the intoken flow and the outtoken flow of an event w.r.t. a canonical token flow function, respect the weight function and the initial marking of (N, m_0). This property is called *token flow property* (Fig. 4).

Definition 2 (*Token Flow Property*). Let $W((l(v_0), p)) = m_0(p)$ for each place $p \in P$. Then $\text{lpo} = (V, \prec, l)$ fulfills the *token flow property (TFP) w.r.t.* (N, m_0) if, for all $p \in P$, there is a token flow function $\chi_p : \prec^0 \to \mathbb{N}$ satisfying (**IN**) $\forall v \in V : \text{In}(v, \chi_p) = W((p, l(v)))$ and (**OUT**) $\forall v' \in V^0 : \text{Out}(v', \chi_p) \leqslant W((l(v'), p))$.

If, for some fixed place p, there is such a token flow function χ_p, we also say that lpo fulfills the *TFP w.r.t.* p.

Theorem 3 (*[1]*). *An LPO is executable if, and only if, it fulfills the token flow property.*

2.3. Polynomial algorithms

In this subsection, we will present two polynomial approaches to test a given LPO for the TFP. While the second one has a faster runtime, the first one allows a better fault analysis in case an LPO fails to be an execution. Both algorithms are based on flow theory (see, for example, [24]).

2.3.1. Iterative procedure

To describe the algorithm, which was also presented in the conference paper [1], we fix a marked p/t-net (N, m_0), $N = (P, T, F, W)$, an LPO $\text{lpo} = (V, \prec, l)$ with $l(V) = T$, a 0-extension $\text{lpo}^0 = (V^0, \prec^0, l^0)$ of lpo and a place p. The

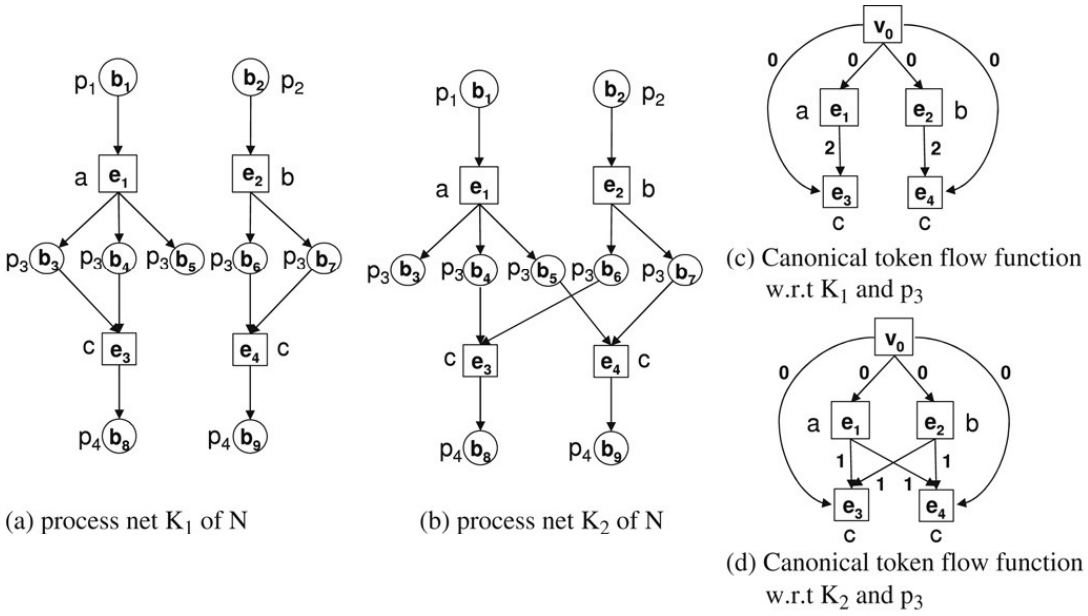


Fig. 3. Two processes (parts (a) and (b)) of the p/t-net N from Fig. 1(a) and the two corresponding runs (parts (c) and (d)) with annotated canonical token flow function w.r.t. the place p_3.

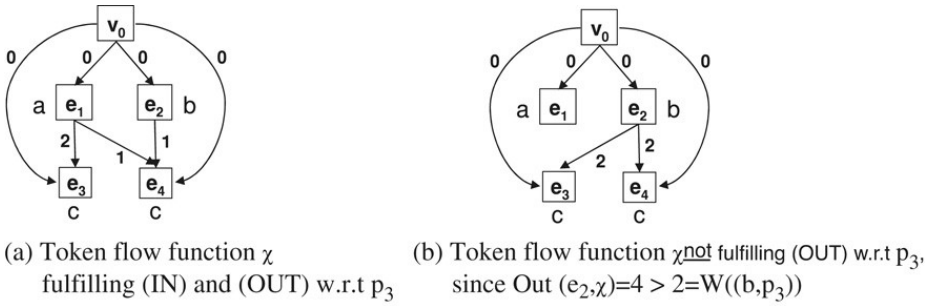


Fig. 4. LPOs fulfilling the TFP (part (a)) and not fulfilling the TFP (part (b)) w.r.t. the p/t-net N from Fig. 1(a).

algorithm is based on an iterative procedure w.r.t. a fixed total ordering $V^0 = \{v_0, v_1, \ldots, v_n\}$ with $v_i \prec^0 v_j \Rightarrow i < j$. In the case lpo fulfills the token flow property w.r.t. p, the algorithm constructs a token flow function χ_p fulfilling (**IN**) and (**OUT**) w.r.t. p. In the case that lpo does not fulfill the TFP w.r.t. p, a prefix of lpo is computed,

- which is enabled w.r.t. p,
- and whose subsequent cut of events represents a multi-set of transitions which are not concurrently enabled w.r.t. p after the occurrence of the prefix.

This proves the correctness of the algorithm. Moreover, the computation of such prefixes allows a detailed fault analysis.

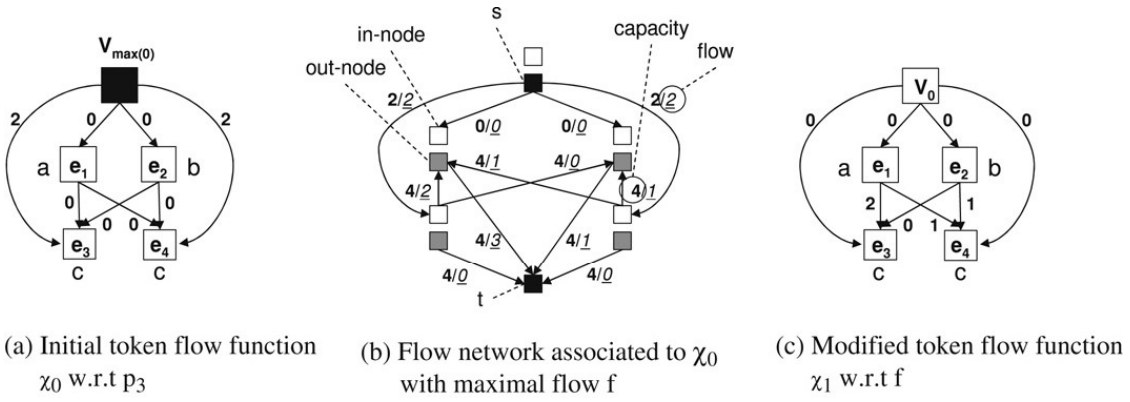
The algorithm starts with an *initial token flow function* χ_0^p fulfilling (**IN**) for all events and iteratively modifies this token flow function in such a way that (**OUT**) is satisfied for a growing set of nodes, while (**IN**) remains preserved for all nodes (w.r.t. the fixed place p). We denote by χ_i^p the token flow function computed after i subsequent modifications of χ_0 and by $\max(\chi_i^p)$ the greatest index k such that χ_i^p satisfies (**OUT**) w.r.t. the events $v_0, \ldots, v_{k-1}$. If p is clear from the context, we write for short $\chi_i = \chi_i^p$ and $\max(i) = \max(\chi_i^p)$. χ_i is modified by a polynomial procedure **Mod**(χ_i) which returns a token flow function χ_{i+1} with the following formal properties:

(Mod1) $\forall v' \in V:\ \mathrm{In}(v', \chi_{i+1}) = \mathrm{In}(v', \chi_i)$.
(Mod2) $\forall k < \max(i):\ Out(v_k, \chi_{i+1}) \leqslant W((l(v_k), p))$.
(Mod3) $\mathrm{Out}(v_{\max(i)}, \chi_{i+1}) \leqslant \mathrm{Out}(v_{\max(i)}, \chi_i)$.

Notice that an initial token flow function always exists. For example define $\chi_0 :\prec^0 \to \mathbb{N}$ by $\chi_0((v, v')) = W((p, l(v')))$ for $v = v_0$ and $\chi_0((v, v')) = 0$ else (Fig. 5 (a)). It is easy to see, that χ_0 fulfills property (**Tff**). The algorithm terminates, if either

(T1) χ_i fulfills property (**OUT**) for all nodes — in this case χ_i is a token flow function showing that lpo fulfills the TFP w.r.t. the considered place p, or
(T2) $\max(i) = \max(i-1)$ — in this case we will prove in Theorem 11 that lpo is not enabled w.r.t. (N, m_0).

Algorithm 1 summarizes the described technique.



(a) Initial token flow function χ_0 w.r.t p_3 (b) Flow network associated to χ_0 with maximal flow f (c) Modified token flow function χ_1 w.r.t f

Fig. 5. Application of **Mod**(χ_0) for the p/t-net N shown in Fig. 1 (a). Observe that χ_1 fulfills the token flow property w.r.t. p_3.

Algorithm 1 (*Tests Whether* lpo *Fulfills The TFP w.r.t. p*).
Step 1: Compute an initial token function χ_0^p and set $i = 0$ ($i \in \mathbb{N}$).
Step 2: Repeat as long as χ_i^p *does not fulfill* (**OUT**) *and* $\max(\chi_i^p) > \max(\chi_{i-1}^p)$: Compute $\chi_{i+1}^p = \mathbf{Mod}(\chi_i^p)$ and increase i by one.
Step 3: Return *true*, if and only if χ_i^p fulfills (**OUT**).

This algorithm has to be applied for every place $p \in P$. χ_i^p fulfills (**OUT**) if, and only if, $\max(i) = n + 1$. Since v_n always satisfies (**OUT**), **Mod**() is repeated, at most, n times.

The modification of χ_i is based on flow theory.

A *flow network* is a tuple (G, c, s, t), where $G = (V, E)$ is a directed graph, $c : E \to \mathbb{N}$ is the *capacity function*, $s \in V$ is the unique node with ${}^\bullet s = \emptyset$ called *source* and $t \in V$ is the unique node with $t^\bullet = \emptyset$ called *sink*. For a compact representation, we extend the capacity function c to pairs of nodes $(x, y) \in (V \times V) \setminus E$ by $c((x, y)) = 0$.

A *flow* f in a flow network is a function $f : E \to \mathbb{N}$ satisfying $\forall e \in E : f(e) \leqslant c(e)$ (capacity constraint) and $\forall v \in V \setminus \{s, t\} : \sum_{v' \in {}^\bullet v} f((v', v)) = \sum_{v' \in v^\bullet} f((v, v'))$ (flow conservation property). The *value* $|f| = \sum_{v' \in s^\bullet} f((s, v'))$ of a flow f is the outgoing flow of the source. It can be equivalently computed as the ingoing flow of the sink. A *maximal flow* is a flow with maximal value among all flows.

The *Maximal Flow Problem* is to compute the value of a maximal flow in a flow network. This problem can be solved in polynomial time by explicit construction of a maximal flow. The best algorithms (based on different methods) have time complexity $O(n^3)$ [25,26], $O(ne \log(n^2/e))$ [26] and $O(ne + n^2(\log c^*)^{1/2})$ [27], where n is the number of nodes, e the number of arcs and c^* the maximal capacity of an arc of the flow network.

Without loss of generality, in this paper we only consider flows such that there is no cycle with positive flow in the flow network.

The aim of the modification of χ_i is to decrease the outtoken flow of $v_{\max(i)}$. This can be done by decreasing the token flow on some edge $(v_{\max(i)}, v)$. Since this decreases the intoken flow of v, we have to increase the token flow on another ingoing edge (v', v) of v (by the same amount) in order to ensure (**IN**). This in turn increases the outtoken flow of v', i.e. we have redistributed outoken flow from $v_{\max(i)}$ to v'. If this contradicts (**Mod2**), we can try the same for v' and so on.

We will represent the amount of change of χ_i by a flow in an appropriate flow network associated to lpo and χ_i. In a natural way, the flow conservation property will ensure that intoken and outoken flows are not changed on "intermediate" nodes. The basic idea of the construction is that the flow computed so far, can still be increased if, and only if, χ_i can still be modified, decreasing the outtoken flow of $v_{\max(i)}$ — i.e. the minimal possible outtoken flow of $v_{\max(i)}$ can be computed through a maximal flow in the flow network.

Edges in lpo are represented in the flow network in original and in reverse order. Flow on edges in original order will be subtracted from the token flow given by χ_i, flow on edges in reverse order will be added. On edges of lpo with positive value of χ_i, token flow can be subtracted. Therefore, such edges are also drawn in the flow network. Besides, on all edges, token flow can be added. Therefore, all edges of lpo are drawn in reverse order in the flow network. In order to preserve the properties (**Tff**), (**IN**) and (**OUT**), each event v of lpo is split into a node (v, out) (reflecting the outtoken flow of v) and a node (v, in) (reflecting the intoken flow of v) of the flow network. The node $(v_{\max(i)}, \text{out})$ serves as the source of the flow network.

Definition 4 (*Associated Flow Network*). Denote the *residue of* v *w.r.t.* χ_i $R(v, \chi_i) = W((l(v), p)) - \text{Out}(v, \chi_i)$. The flow network (G, c, s, t), $G = (W, E)$, *associated to* lpo *and* χ_i is defined by $W = (V \times \{\text{in}, \text{out}\}) \cup \{t\}$, $s = (v_{\max(i)}, \text{out})$, $E = E_{\text{lpo}} \cup E_{\text{lporev}} \cup E_{\text{upper}} \cup E_{\text{lower}}$ and $c : E \to \mathbb{N}$, where

$$E_{\text{lpo}} = \{((v_j, \text{out}), (v_l, \text{in})) \mid j \leqslant \max(i),\ \chi_i((v_j, v_l)) > 0\},$$
$$E_{\text{lporev}} = \{((v_l, \text{in}), (v_j, \text{out})) \mid j \neq \max(i),\ v_j \prec^0 v_l\},$$
$$E_{\text{upper}} = \{((v_j, \text{out}), t) \mid j > \max(i)\},$$
$$E_{\text{lower}} = \{((v_j, \text{out}), t) \mid j < \max(i)\},$$
$$c(e) = \chi_i((v_j, v_l)) \text{ if } e = ((v_j, \text{out}), (v_l, \text{in})) \in E_{\text{lpo}},$$
$$c(e) = \text{Out}(v_{\max(i)}, \chi_i) \text{ if } e = ((v_l, \text{in}), (v_j, \text{out})) \in E_{\text{lporev}},$$
$$c(e) = \text{Out}(v_{\max(i)}, \chi_i) \text{ if } e = ((v_j, \text{out}), t) \in E_{\text{upper}},$$
$$c(e) = R(v_j, \chi_i) \text{ if } e = ((v_j, \text{out}), t) \in E_{\text{lower}}.$$

As mentioned, a flow on edges in E_{lpo} is subtracted from χ_i. Therefore, the flow through such edges is bounded by the value of χ_i. If there is a non-zero flow, the outtoken flow of $v_{\max(i)}$ is decreased by this flow.

A flow on edges in E_{lporev} is added to χ_i. The capacity $\text{Out}(v_{\max(i)}, \chi_i)$ on such edges is chosen not to restrict the maximal possible flow. An important characterization of maximal flows considers so-called minimal flow cuts. A *flow cut* is a pair of sets $X, Y \subseteq V$ with $X \cup Y = V, X \cap Y = \emptyset, s \in X$ and $t \in Y$. The *capacity of a flow cut* is $c(X, Y) = \sum_{x \in X,\, y \in Y,\, x \to y} c((x, y))$. The famous *maximal flow-minimal flow cut theorem* states that the maximum flow in a flow network equals the minimum capacity of a flow cut in this flow network.[5] The capacity $\text{Out}(v_{\max(i)}, \chi_i)$ is the capacity of the flow cut $(\{s\}, W \setminus \{s\})$.

If, for an event v_j with $j \neq \max(i)$ there is no flow from (v_j, out) to the sink t, then by construction, and from the properties of flows, we get that these modifications of χ_i do not change the intoken flow or the outtoken flow of v_j. If there is a flow from (v_j, out) to t, the outtoken flow of v_j is increased. If $j > \max(i)$ (flow on an edge in E_{upper}), such edges need no restrictive capacity bound. On the other hand, if $j < \max(i)$ (flow on an edge in E_{lower}), the flow is restricted by $R(v_j, \chi_i)$ in order not to violate (**OUT**). Fig. 5 (b) shows an associated flow network.

We now formally define how to modify χ_i by a flow in the associated flow network.

Definition 5 (*Modified Token Flow Function*). For a flow f in (G, c, s, t), define the token flow function χ_f *modifying* χ_i *w.r.t.* f as follows:

- $\chi_f((v_j, v_l)) = \chi_i((v_j, v_l)) - f(((v_j, \text{out}), (v_l, \text{in})))$ if $((v_j, \text{out}), (v_l, \text{in})) \in E_{\text{lpo}}$,
- $\chi_f((v_j, v_l)) = \chi_i((v_j, v_l)) + f((v_l, \text{in}), (v_j, \text{out}))$ if $((v_l, \text{in}), (v_j, \text{out})) \in E_{\text{lporev}}$,
- $\chi_f((v, v')) = \chi_i((v, v'))$ else.

The following lemma shows that the presented modification yields the intended properties.

Lemma 6. *Let f be a flow in (G, c, s, t). Then χ_f satisfies* **(Mod1)**–**(Mod3)** *with* $\text{Out}(v_{\max(i)}, \chi_f) = \text{Out}(v_{\max(i)}, \chi_i) - |f|$.

Proof. Denote $\prec_{\text{lpo}} = \{(v, v') \in \prec^0 \mid ((v, \text{out}), (v', \text{in})) \in E_{\text{lpo}}\}$ and $\prec_{\text{lporev}} = \{(v, v') \in \prec^0 \mid ((v', \text{in}), (v, \text{out})) \in E_{\text{lporev}}\}$. Property **(Mod1)** follows from the following computation for $(v', \text{in}) \in W$, using the second defining property of flows (ingoing and outgoing flow of each node coincide):

$$\begin{aligned}\sum_{v \prec_{\text{lpo}} v'} f(((v, \text{out}), (v', \text{in}))) &= \sum_{\mu \in {}^\bullet(v', \text{in})} f((\mu, (v', \text{in})))\\ &= \sum_{\mu \in (v', \text{in})^\bullet} f(((v', \text{in}), \mu))\\ &= \sum_{v \prec_{\text{lporev}} v'} f(((v', \text{in}), (v, \text{out}))).\end{aligned}$$

We get $\text{In}(v', \chi_f) = \text{In}(v', \chi_i)$ for $v' \in V$ because $\text{In}(v', \chi_f) = \text{In}(v', \chi_i) + \sum_{v \prec_{\text{lporev}} v'} f(((v', \text{in}), (v, \text{out}))) - \sum_{v \prec_{\text{lpo}} v'} f(((v, \text{out}), (v', \text{in})))$. Analogously, we deduce **(Mod2)** from the following computation for $(v, \text{out}) \in W \setminus \{(v^0, \text{out})\}$:

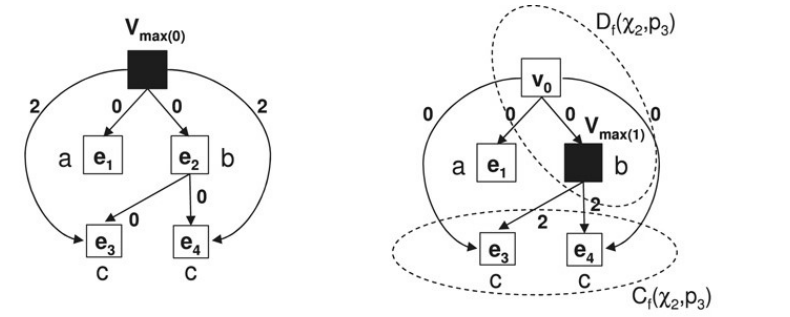
$$\sum_{v \prec_{\text{lporev}} v'} f(((v', \text{in}), (v, \text{out}))) = f(((v, \text{out}), t)) + \sum_{v \prec_{\text{lpo}} v'} f(((v, \text{out}), (v', \text{in}))),$$

For $k < \max(i)$ this implies $\text{Out}(v_k, \chi_f) = \text{Out}(v_k, \chi_i) + \sum_{v' \prec_{\text{lporev}} v_k} f(((v', \text{in}), (v_k, \text{out}))) - \sum_{v_k \prec_{\text{lpo}} v'} f(((v_k, \text{out}), (v', \text{in}))) = \text{Out}(v_k, \chi_i) + f(((v_k, \text{out}), t)) \leqslant \text{Out}(v_k, \chi_i) + R(v_k, \chi_i) = W((l(v_k), p))$. We will re-use the equation

$$(*) \qquad \text{Out}(v_k, \chi_f) = \text{Out}(v_k, \chi_i) + f(((v_k, \text{out}), t))$$

in the proof of Lemma 8 (ii).

[5] We use the term *flow cut* here, instead of the usual term *cut* in order to get not confused with cuts in partial orders.



(a) Initial token flow function χ_0 w.r.t p_3

(b) Modified token flow function $\chi_{1=\chi 2.}$ It does not fulfil (OUT) w.r.t p_3.

Fig. 6. An LPO which is not an execution of the p/t-net N shown in Fig. 1(a) with computed token flow functions and critical coset.

With the definition of $|f|$ we get:

$$\begin{aligned}\mathrm{Out}(v_{\max(i)}, \chi_f) &= \mathrm{Out}(v_{\max(i)}, \chi_i) - \sum_{v_{\max(i)} \prec_{\mathrm{lpo}} v'} f(((v_{\max(i)}, \mathrm{out}), (v', \mathrm{in})))\\ &= \mathrm{Out}(v_{\max(i)}, \chi_i) - \sum_{\mu \in (v_{\max(i)}, \mathrm{out})^\bullet} f(((v_{\max(i)}, \mathrm{out}), \mu))\\ &= \mathrm{Out}(v_{\max(i)}, \chi_i) - |f|.\end{aligned}$$

The function χ_f is a token flow function, because **(Mod1)** implies (**Tff**).

We are now able to formally introduce the procedure **Mod**(χ_i):

Algorithm 2 (*Procedure* **Mod**$(\chi_i) = \chi_{i+1}$).
Step 1: Compute the flow network (G, c, s, t) associated to lpo and χ_i.
Step 2: Compute a maximal flow f in (G, c, s, t).
Step 3: Return $\chi_{i+1} = \chi_f$ (Fig. 5 (c)).

The final verification procedure Algorithm 3 applies Algorithm 1 to each place $p \in P$ with integrated procedure **Mod**().

Algorithm 3 (*Tests. If* lpo *is an Execution of* (N, m_0)).
Step 1: Repeat for all places $p \in P$:
Step 1.1: Compute an initial token function χ_0^p and set $i = 0$ $(i \in \mathbb{N})$.
Step 1.2: Repeat as long as χ_i^p *does not fulfill* (**OUT**) *and* $\max(\chi_i^p) > \max(\chi_{i-1}^p)$:
Step 1.2.1: Compute the flow network (G, c, s, t) associated to lpo and χ_i^p.
Step 1.2.2: Compute a maximal flow f in (G, c, s, t).
Step 1.2.3: Compute χ_f, set $\chi_{i+1}^p = \chi_f$ and increase i by one.
Step 2: Return *true* if, and only if, χ_i^p fulfills (**OUT**) for each $p \in P$.

It remains to prove the correctness of this algorithm. Lemma 6 says that lpo fulfills the TFP w.r.t. the place p, if the loop of Algorithm 1 terminates because χ_i satisfies (**OUT**) (case **(T1)**). Thus, if Algorithm 3 returns *true*, lpo is an execution. Algorithm 3 returns *false*, if the loop in Algorithm 1 terminates for some place because $\max(i) = \max(i-1)$ for some i (case **(T2)**). In this case we show that lpo is not an execution, using the equivalent characterization of executions as enabled LPOs. That means we construct a cut C of lpo such that $m_0(p) + \sum_{v \in V \wedge v \prec C}(W((l(v), p)) - W((p, l(v)))) < \sum_{v \in C} W((p, l(v)))$ (Fig. 6(b)).

This cut C is constructed in several steps. First we define the set of nodes $D_f(\chi_i, p)$ which turns out to define a prefix enabled w.r.t. p. Next we define the set of nodes $C_f(\chi_i, p)$ which turns out be the coset having $D_f(\chi_i, p)$ as its set of smaller events. We will prove, that after the occurrence of the prefix given by $D_f(\chi_i, p)$ the step given by $C_f(\chi_i, p)$ is not enabled. Finally we extend the coset $C_f(\chi_i, p)$ to the cut $C(\chi_i, p)$ with the same set of smaller events. Since $C_f(\chi_i, p)$ is not enabled, $C(\chi_i, p)$ is also not enabled, i.e. $C(\chi_i, p)$ will be the searched cut.

Definition 7 (*Critical Coset (Cut)*). Let f be a maximal flow of the network associated to lpo and χ_i. Assume that χ_f does not fulfill (**OUT**) for the node $v_{\max(i)}$. Let $D_f(\chi_i, p)$ be the set of all nodes $v \in V^0$ such that there exists a sequence of nodes $\sigma(v) = v^0 w^1 v^1 \ldots w^k v^k$ with $v^0 = v_{\max(i)}$ and $v^k = v$ satisfying (**C1**) $\forall j \neq m : w^j \neq w^m \wedge v^j \neq v^m$ and (**C2**) $\forall j : \chi_f(v^j, w^{j+1}) > 0 \wedge v^j \prec^0 w^j$. Then the set

$$C_f(\chi_i, p) = \{w \in V \setminus D_f(\chi_i, p) \mid \exists v \in D_f(\chi_i, p) : \chi_f((v, w)) > 0\}$$

is called *critical coset (w.r.t. χ_i and p)*. The set

$$C(\chi_i, p) = \{w \in V \setminus D_f(\chi_i, p) \mid (v \prec^0 w) \implies (v \in D_f(\chi_i, p))\}$$

is called *critical cut (w.r.t. χ_i and p)*.

For a node $v \in D_f(\chi_i, p)$ and a corresponding sequence $\sigma(v) = v^0 w^1 v^1 \dots w^k v^k$ it holds $\forall j \leqslant k : \ v^j \in D_f(\chi_i, p)$ and $w^j \notin D_f(\chi_i, p) \iff w^j \in C_f(\chi_i, p) \quad (1 \leqslant j \leqslant k)$.

We first show that $D_f(\chi_i, p)$ defines a prefix enabled w.r.t. p and that $C_f(\chi_i, p)$ is a coset having $D_f(\chi_i, p)$ as its set of smaller events. Moreover, the next lemma prepares the computation of the marking of p after the occurrence of the prefix.

For this, we use the characterization of maximal flows through so-called flow augmenting paths. Some of the maximal flow algorithms are based on the idea of iteratively increasing the flow along such *flow augmenting paths* (starting with the 0-flow). This idea was first proposed in [28] (leading to a pseudo-polynomial $O(ef^*)$-algorithm, where f^* denotes the value of a maximal flow, and improved, for example, in [25], where an $O(n^3)$-algorithm is presented).

Flow augmenting paths are defined in a so-called *residual network* (G_f, c_f, s, t), $G_f = (V, E_\rightarrow)$, *of* (G, c, s, t) *w.r.t. a flow* f, defined by the set of edges $E_\rightarrow = \{(v, v') \in V \times V \mid (v, v') \in E \vee (v', v) \in E\}$ and the *residual capacity function* $c_f : E_\rightarrow \to \mathbb{N}$ given by $c_f((v, v')) = c((v, v')) - f((v, v'))$ if $(v, v') \in E \wedge (v', v) \notin E$, $c_f((v, v')) = f((v', v))$ if $(v, v') \notin E \wedge (v', v) \in E$ and by $c_f((v, v')) = c((v, v')) - (f((v, v')) - f((v', v)))$ if $(v, v'), (v', v) \in E$. A *flow augmenting path of* N *w.r.t.* f is a simple path $v_0 \dots v_n$ from $s = v_0$ to $t = v_n$ in $(V, E_\rightarrow)$ with $c_f((v_{i-1}, v_i)) > 0$ for $i \in \{1, \dots, n\}$.

In [28], it is proven that there is no flow augmenting path of the flow network w.r.t. f if, and only if, f is maximal. Moreover, it is shown there that in flow networks with integer capacities, there are always integer maximal flows.

Lemma 8. *Let* f *be a maximal flow of the network associated with* lpo *and* χ_i. *Assume that* χ_f *does not fulfill* (**OUT**) *for the node* $v_{\max(i)}$. *It holds:*

(i) $v_j \in D_f(\chi_i, p) \implies j \leqslant \max(i)$.
(ii) $(v_j \in D_f(\chi_i, p) \wedge j \neq \max(i)) \implies R(v_j, \chi_f) = 0$.
(iii) $(\exists w \in C_f(\chi_i, p) : \ v \prec^0 w) \iff v \in D_f(\chi_i, p)$.

Proof. To prove (i) and (ii) we assume the converse, and deduce that, then, there is a flow augmenting path w.r.t. f in the associated flow network — this is a contradiction to the maximality of f.

Since by assumption $|f| < \mathrm{Out}(v_{\max(i)}, \chi_i)$ (and since there is no positive flow along cycles) also $f(e) < \mathrm{Out}(v_{\max(i)}, \chi_i)$ for each edge e.

ad (i): Let $v_j \in D_f(\chi_i, p)$, $\sigma(v_j) = v^0 w^1 v^1 \dots w^k v^k$ with $j > \max(i)$ and m be the smallest index satisfying $v^m = v_l$ for $l > \max(i)$. We claim that, then, $(v^0, \mathrm{out})(w^1, \mathrm{in})(v^1, \mathrm{out}) \dots (w^m, \mathrm{in})(v^m, \mathrm{out})t$ is a flow augmenting path w.r.t. f in the associated flow network. To prove this, we must show that $(v^0, \mathrm{out})(w^1, \mathrm{in})(v^1, \mathrm{out}) \dots (w^m, \mathrm{in})(v^m, \mathrm{out})t$ is a path in the residual network (G_f, c_f, s, t), $G_f = (W, E_f)$, of (G, c, s, t) w.r.t. f satisfying

- $c_f(((v^{l-1}, \mathrm{out}), (w^l, \mathrm{in}))) > 0 \ (1 \leqslant l \leqslant m)$,
- $c_f(((w^l, \mathrm{in}), (v^l, \mathrm{out}))) > 0 \ (1 \leqslant l \leqslant m)$,
- $c_f(((v^m, \mathrm{out}), t)) > 0$.

From the definitions we get $((w^l, \mathrm{in}), (v^l, \mathrm{out})) \in E_{\mathrm{lporev}}$, i.e.

$$\begin{aligned} c_f(((w^l, \mathrm{in}), (v^l, \mathrm{out}))) &\geqslant c(((w^l, \mathrm{in}), (v^l, \mathrm{out}))) - f(((w^l, \mathrm{in}), (v^l, \mathrm{out}))) \\ &= \mathrm{Out}(v^0, \chi_i) - f(((w^l, \mathrm{in}), (v^l, \mathrm{out}))) > 0. \end{aligned}$$

Moreover, we get $((v^{l-1}, \mathrm{out}), (w^l, \mathrm{in})) \in E_{\mathrm{lpo}}$, i.e.

$$\begin{aligned} c_f(((v^{l-1}, \mathrm{out}), (w^l, \mathrm{in}))) &\geqslant c(((v^{l-1}, \mathrm{out}), (w^l, \mathrm{in}))) - f(((v^{l-1}, \mathrm{out}), (w^l, \mathrm{in}))) \\ &= \chi_i((v^{l-1}, w^l)) - f(((v^{l-1}, \mathrm{out}), (w^l, \mathrm{in}))) \\ &= \chi_f((v^{l-1}, w^l)) > 0. \end{aligned}$$

Finally, $((v^m, \mathrm{out}), t) \in E_{\mathrm{upper}}$, i.e.

$$\begin{aligned} c_f(((v^m, \mathrm{out}), t)) &\geqslant c(((v^m, \mathrm{out})), t) - f(((v^m, \mathrm{out}), t)) \\ &= \mathrm{Out}(v^0, \chi_i) - f(((v^m, \mathrm{out}), t)) > 0. \end{aligned}$$

ad (ii): Let $v_j \in D_f(\chi_i, p)$ and $\sigma(v_j) = v^0 w^1 v^1 \dots w^k v^k$ with $j \neq \max(i)$ and $R(v_j, \chi_f) \neq 0$. According to (i) we have $j < \max(i)$. Since χ_f satisfies **(Mod2)**, it follows $R(v_j, \chi_f) > 0$. We claim that $(v^0, \mathrm{out})(w^1, \mathrm{in})(v^1, \mathrm{out}) \dots (w^k, \mathrm{in})(v^k, \mathrm{out})t$ is a flow augmenting path w.r.t. f in the associated flow network. We show that

- $c_f(((v^{l-1}, \mathrm{out}), (w^l, \mathrm{in}))) > 0 \ (1 \leqslant l \leqslant k)$,
- $c_f(((w^l, \mathrm{in}), (v^l, \mathrm{out}))) > 0 \ (1 \leqslant l \leqslant k)$,
- $c_f(((v^k, \mathrm{out}), t)) > 0$.

As above, we deduce $c_f(((w^l, \text{in}), (v^l, \text{out}))) > 0$ and $c_f(((v^{l-1}, \text{out}), (w^l, \text{in}))) > 0$. Finally, we get (the fourth equation follows from the computation $(*)$ in the proof of Lemma 6)

$$\begin{aligned} c_f(((v^k, \text{out}), t)) &\geqslant c(((v^k, \text{out}), t)) - f(((v^k, \text{out}), t)) \\ &= R(v^k, \chi_i) - f((v^k, \text{out}), t) \\ &= W((l(v^k), p)) - \text{Out}(v^k, \chi_i) - f((v^k, \text{out}), t) \\ &= W((l(v^k), p)) - \text{Out}(v^k, \chi_f) \\ &= R(v^k, \chi_f) > 0. \end{aligned}$$

ad (iii) $\Longrightarrow$: Let $w \in C_f(\chi_i, p)$ with $v \prec^0 w$. We construct a sequence $\sigma(v) = v_{\max(i)} \dots v$ fulfilling (**C1**) and (**C2**). By the definition of $C_f(\chi_i, p)$ there is a node $v' \in D_f(\chi_i, p)$ with $\chi_f((v', w)) > 0$. Let $\sigma(v') = v_{\max(i)} w^1 v^1 \dots w^k v^k$. In the cases $v = v'$ or $v = v^j$ for $j \in \{0, \dots, k\}$ it follows $v \in D_f(\chi_i, p)$. We distinguish the following remaining cases:

- $(\exists j \in \{0, \dots, k\} : \ w^j = w)$: $v_{\max(i)} w^1 v^1 \dots w^j v$ satisfies (**C1**) and (**C2**).
- $(\forall j \in \{0, \dots, k\} : \ w^j \neq w)$: $v_{\max(i)} w^1 v^1 \dots w^k v' w v$ satisfies (**C1**) and (**C2**).

ad (iii) $\Longleftarrow$: Let $v \in D_f(\chi_i, p)$ and $\sigma(v) = v_{\max(i)} w^1 v^1 \dots w^k v^k$. We will find $w \in C_f(\chi_i, p)$ with $v \prec^0 w$. For this, we distinguish the following cases:

- $v = v_{\max(i)}$: By assumption, it holds $v_{\max(i)} \prec^0 C_f(\chi_i, p)$ since $v_{\max(i)}$ has positive outtoken flow.
- $w^k \in C_f(\chi_i, p)$: $v = v^k \prec^0 w^k \in C_f(\chi_i, p)$.
- $w^k \in D_f(\chi_i, p)$: Let $\overline{v}$ be a maximal node in the set $\{v' \in D_f(\chi_i, p) \mid v \prec^0 v'\}$ w.r.t. $\prec^0$ (the set is not empty since w^k is one of its elements). Let $\sigma(\overline{v}) = v_{\max(i)} \overline{w}^1 \overline{v}^1 \dots \overline{w}^l \overline{v}^l$ satisfy (**C1**) and (**C2**). Then $\overline{w}^l \notin D_f(\chi_i, p)$ (otherwise $\overline{v}$ would not be maximal) and thus $v \prec^0 \overline{v} \prec^0 \overline{w}^l \in C_f(\chi_i, p)$.

Property (iii) of the last lemma directly implies that $C_f(\chi_i, p)$ is a coset and that $D_f(\chi_i, p) \subseteq V$ defines a prefix. From Property (i) we easily deduce that the prefix defined by $D_f(\chi_i, p)$ is enabled.

The following straightforward lemma shows that $C(\chi_i, p)$ is the extension of the coset $C_f(\chi_i, p)$ to a cut with the same set of smaller events.

Lemma 9. *It holds:*

(i) $C_f(\chi_i, p) \subseteq C(\chi_i, p)$.
(ii) $v \prec^0 C_f(\chi_i, p) \Longleftrightarrow v \prec^0 C(\chi_i, p)$.
(iii) $C(\chi_i, p)$ *is a cut.*

Proof. ad (i): Let $w \in C_f(\chi_i, p)$ and $v' \prec^0 w$. We have to show that $v' \in D_f(\chi_i, p)$. For this, we construct a sequence $\sigma(v') = v_{\max(i)} \dots v'$ satisfying (**C1**) and (**C2**). By definition, there is a node $v \in D_f(\chi_i, p)$ with $\chi_f((v, w)) > 0$. Let $\sigma(v) = v_{\max(i)} w^1 v^1 \dots w^k v^k$. If $v = v^j$ for some j then clearly $v' \in D_f(\chi_i, p)$. Let $v \neq v^j$ for all j: If $w = w^j$ for some j then we set $\sigma(v') = v_{\max(i)} w^1 v^1 \dots w^j v'$, otherwise we set $\sigma(v') = v_{\max(i)} w^1 v^1 \dots w^k v^k w v'$.

ad (ii): According to Lemma 8 (iii) it holds $v \in D_f(\chi_i, p) \implies v \prec^0 C_f(\chi_i, p)$. Therefore, it is enough to show that $v \prec^0 C_f(\chi_i, p) \implies v \prec^0 C(\chi_i, p) \implies v \in D_f(\chi_i, p)$. The first implication follows from $C_f(\chi_i, p) \subseteq C(\chi_i, p)$, the second one follows from the definition of $C(\chi_i, p)$.

ad (iii): By definition $C(\chi_i, p)$ is a coset. It remains to show that $C(\chi_i, p)$ is maximal. Let $v \notin C(\chi_i, p)$. We will prove that, then, there is a node $w \in C(\chi_i, p)$ with $v \prec^0 w$ or $w \prec^0 v$. We distinguish the following cases:

- $v \in D_f(\chi_i, p)$: From (i) and Lemma 8 (iii) we deduce $v \prec^0 w$ for some $w \in C(\chi_i, p)$.
- $v \notin D_f(\chi_i, p)$: The set of nodes $v' \prec^0 v$ with $v' \in D_f(\chi_i, p)$ is not empty because $v_0 \in D_f(\chi_i, p)$ according to Lemma 8 (iii). Since $v \notin C(\chi_i, p)$, by the definition of $C(\chi_i, p)$ there must be a node $v' \prec^0 v$ with $v' \notin D_f(\chi_i, p)$. Let m be the smallest index with $v_m \notin D_f(\chi_i, p)$ and $v_m \prec^0 v$. Then $v_m \in C(\chi_i, p)$ by the definition of $C(\chi_i, p)$ (otherwise there would be a smaller index).

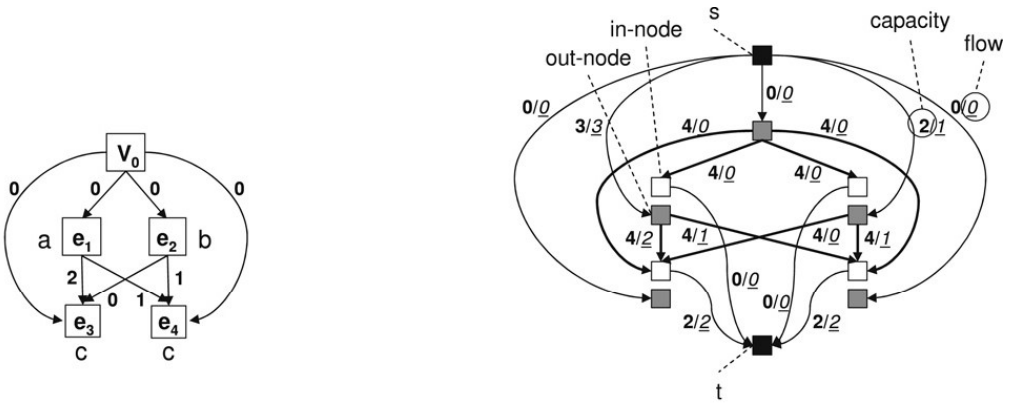
We finally compute that, after occurrence of the prefix defined by $D_f(\chi_i, p)$, the step given by the cut $C(\chi_i, p)$ is not enabled.

Lemma 10. *It holds for* $C = C(\chi_i, p)$*:*

$$m_0(p) + \sum_{v \prec C} (W((l(v), p)) - W((p, l(v)))) - \sum_{v \in C} W((p, l(v))) < 0.$$

Proof. We first consider the coset $C = C_f(\chi_i, p)$. The token flow function χ_f has the following properties:

- $W((l(v_{\max(i)}), p)) < \text{Out}(v_{\max(i)}, \chi_f) = \sum_{v_{\max(i)} \prec^0 v'} \chi_f((v_{\max(i)}, v'))$.



(a) Enabled LPO with token flow function χ fulfilling (IN) and (OUT) w.r.t. p_3

(b) Associated flow network with maximal flow f corresponding to χ

Fig. 7. Associated flow network with maximal flow corresponding to a token flow function fulfilling (**IN**) and (**OUT**) w.r.t the p/t-net N shown in Fig. 1(a).

- $\forall v \in C_f(\chi_i, p) \cup D_f(\chi_i, p) : W((p, l(v))) = \mathrm{In}(v, \chi_f) = \sum_{v' \prec^0 v} \chi_f((v', v))$.
- $\forall v \in D_f(\chi_i, p) \setminus \{v_{\max(i)}\} : W((l(v), p)) = \mathrm{Out}(v, \chi_f) = \sum_{v \prec^0 v'} \chi_f((v, v'))$ (Lemma 8 (ii)).

With $m_0(p) = W((l(v_0), p))$ it is enough to show

$$m_0(p) + \sum_{v \prec C} (W((l(v), p)) - W((p, l(v)))) - \sum_{v \in C} W((p, l(v)))$$
$$< \sum_{v \prec^0 C} \sum_{v \prec^0 v'} \chi_f((v, v')) - \sum_{v' \prec^0 v} \chi_f((v', v)) \Bigg) - \sum_{v \in C} \sum_{v' \prec^0 v} \chi_f((v', v)) = 0.$$

The inequation is clear by the above considerations. We claim that, in the second sum, each summand $\chi_f((v, v'))$ either (i) equals 0, or (ii) is counted exactly once positively and once negatively. For $(v, v') \in D_f(\chi_i, p) \times (D_f(\chi_i, p) \cup C_f(\chi_i, p))$ case (ii) holds according to Lemma 8 (iii). For $(v, v') \in D_f(\chi_i, p) \times (V^0 \setminus D_f(\chi_i, p))$ with $\chi_f((v, v')) > 0$ we have $v' \in C_f(\chi_i, p)$ by definition. That means (ii) holds in each case (i) does not hold.

Since $C(\chi_i, p)$ extends $C_f(\chi_i, p)$ to a cut with the same set of smaller events, the statement follows.

Theorem 11. *Let f be a maximal flow of the network associated to* lpo *and χ_i^p for some place p. Assume that χ_f does not fulfill* (**OUT**) *for the node $v_{\max(\chi_i^p)}$. Then there is a cut $C \subseteq V$ of* lpo*, such that $m_0(p) + \sum_{v \in V \wedge v \prec C} (W((l(v), p)) - W((p, l(v)))) < \sum_{v \in C} W((p, l(v)))$.*

2.3.2. Direct transformation

In this subsection, we present another polynomial algorithm to test whether an LPO fulfills the TFP. It is proposed in [3] to improve the performance. It is based on a direct transformation of the LPO into a flow network. As for the previous algorithm, in the case that lpo fulfills the TFP, this new algorithm constructs respective token flow functions for every place. Throughout this subsection, we use the same notations as in the last one. For each place p we will construct a flow network (G, c, s, t) associated to lpo (and p) and define a natural number $M(\mathrm{lpo}, p)$ such that lpo fulfills the TFP w.r.t. p if, and only if, the value of a maximum flow in (G, c, s, t) equals $M(\mathrm{lpo}, p)$.

The idea of the construction of (G, c, s, t) is to compute a token flow function satisfying (**IN**) and (**OUT**) (if such a token flow function exists) by a maximal flow in (G, c, s, t), $G = (W, E)$. That means, in particular, that the outtoken flow of a node of lpo equals the flow outgoing from some corresponding node of (G, c, s, t). Also the intoken flow of a node of lpo equals the flow ingoing to some corresponding node of (G, c, s, t). Since, in the flow network, the ingoing flow of a node equals its outgoing flow, one node of lpo is split into two nodes of (G, c, s, t), one to represent the corresponding outtoken flow and the other to represent the corresponding intoken flow. To ensure (**IN**) and (**OUT**), the outgoing flow and the ingoing flow of a node of (G, c, s, t) are restricted by appropriate capacities. An edge of lpo corresponds to an edge of (G, c, s, t) between a node representing the outtoken flow and a node representing the intoken flow. Fig. 7 shows such a flow network.

Definition 12 (*Associated Flow Network*). We denote $M = M(\mathrm{lpo}, p) = \sum_{v \in V} W(p, l(v))$. The flow network (G, c, s, t), $G = (W, E)$, *associated to* lpo *and* p is defined by $W = (V^0 \times \{\mathrm{in}, \mathrm{out}\}) \cup \{s, t\}$, $E = E_s \cup E_{\mathrm{lpo}} \cup E_t$ and $c : E \to \mathbb{N}$, where:

$$E_s = \{(s, (v, \mathrm{out})) \mid v \in V^0\},\ c(e) = W(l(v), p) \text{ if } e = (s, (v, \mathrm{out})) \in E_s,$$
$$E_{\mathrm{lpo}} = \{((v, \mathrm{out}), (v', \mathrm{in})) \mid v \prec^0 v'\},\ c(e) = M \text{ if } e \in E_{\mathrm{lpo}},$$
$$E_t = \{((v, \mathrm{in}), t) \mid v \in V^0\},\ c(e) = W(p, l(v)) \text{ if } e = ((v, \mathrm{in}), t) \in E_t.$$

A flow on an edge $((v, \text{out}), (v', \text{in})) \in E_{\text{lpo}}$ can be interpreted as the number of tokens produced by transition $l(v)$ in place p, which are consumed by transition $l(v')$. That means each flow in (G, c, q, t) has an analogous interpretation as a token flow function $\chi_p : \prec^0 \to \mathbb{N}$ of lpo, defined by

$$\chi_p((v, v')) = f(((v, \text{out}), (v', \text{in}))).$$

χ_p can be considered as a "possible" token flow function of lpo.

Since the flow on an edge $(s, (v, \text{out})) \in E_s$ is, at most, the number of tokens transition $l(v)$ produces in place p, the outgoing flow of a node (v, out) also cannot exceed this number. Therefore χ_p always fulfills property (**OUT**).

Since the flow on an edge $((v, \text{in}), t) \in E_t$ is, at most, the number of tokens transition $l(v)$ consumes from place p, the ingoing flow of a node (v, in) cannot exceed this number. Thus, χ_p fulfills property (**IN**) of the TFP, if the flow on each edge $((v, \text{in}), t) \in E_t$ equals the number of tokens transition $l(v)$ consumes from place p, i.e. equals the capacity on this edge. In this case, χ_p moreover satisfies (**Tff**). That means, if a maximal flow in (G, c, q, t) saturates all edges to the sink (equals $M(\text{lpo}, p)$) this maximal flow defines a token flow function satisfying (**IN**) and (**OUT**) w.r.t. p. The algorithm works as follows:

Algorithm 4 (*Tests, whether* lpo *is an execution of* (N, m_0)).
Step 1: Repeat for each place $p \in P$:
Step 1.1: Compute the flow network (G, c, q, t) associated to lpo and p.
Step 1.2: Compute a maximal flow f_p in (G, c, q, t).
Step 2: Return *true* if, and only if, $|f_p| = M(\text{lpo}, p)$ for each place p.

Theorem 13. *An LPO fulfills the TFP w.r.t. the place p of a marked p/t-net* (N, m_0) *if, and only if, the value of a maximal flow of the associated flow network equals* $M(\text{lpo}, p)$.

Proof. **"if"-part**: Shown in the paragraph before Algorithm 4. **"only if"-part**: Fix a place p and let $\chi_p : \prec^0 \to \mathbb{N}$ be a token flow function fulfilling (**IN**) and (**OUT**) w.r.t. p. We claim that the function $f : E \to \mathbb{N}$, defined as follows, is a maximal flow in (G, c, s, t), $G = (W, E)$, with value $|f| = \sum_{v \in V} W(p, l(v))$:

$$f(e) = \begin{cases} \text{Out}(v, \chi_p) & \text{if } e = (s, (v, \text{out})) \in E_s, \\ \chi_p((v, v')) & \text{if } e = ((v, \text{out}), (v', \text{in})) \in E_{\text{lpo}}, \\ \text{In}(v', \chi_p) & \text{if } e = ((v', \text{in}), t) \in E_t. \end{cases}$$

Directly from this definition, we get that for each node, the ingoing flow equals the outgoing flow defined by f. From (**IN**) and (**OUT**) and the definition of the capacity function, we deduce $f(e) \leqslant c(e)$ for each edge e as follows:

$$\begin{aligned} f((s, (v, \text{out}))) = \text{Out}(v, \chi_p) &\overset{(\text{OUT})}{\leqslant} W(l(v), p) = c((s, (v, \text{out}))) \\ f(((v, \text{out}), (v', \text{in}))) = \chi_p((v, v')) &\overset{(\text{IN})}{\leqslant} W(p, l(v')) \leqslant M(\text{lpo}, p) \\ &= c(((v, \text{out}), (v', \text{in}))) \\ f(((v', \text{in}), t)) = \text{In}(v', \chi_p) &\overset{(\text{IN})}{=} W(p, l(v')) = c(((v', \text{in}), t)) \end{aligned}$$

Moreover, $|f| = \sum_{v' \in V^0} f(((v', \text{in}), t)) = \sum_{v' \in V^0} W(p, l(v')) = M(\text{lpo}, p)$. The flow is maximal, since it saturates the capacity of the cut $(W \setminus \{t\}, \{t\})$ of (G, c, s, t).

2.4. Optimization of the algorithms

In this subsection, we briefly sketch several possibilities to optimize the Algorithms 3 and 4 (see [3] for more details).

The first optimization only concerns Algorithm 3. The computation of the maximal flow f in the flow network associated to a token flow function χ_i and a place p should terminate as soon as $\text{Out}(v_{\max(i)}, \chi_f) = \text{Out}(v_{\max(i)}, \chi_i) - |f| = W((l(v_{\max(i)}), p))$. That means only the excess of the outtoken flow of $v_{\max(i)}$ should be redistributed. In this case the critical node $v_{\max(i)}$ is exactly saturated and thus already satisfies (**OUT**). This can be achieved by bounding the maximal possible flow by the value $R(v_{\max(i)}, \chi_i)$. This bound can directly be implemented into the maximal flow algorithm by adding a new source and appropriately restricting the ingoing flow of the old source $(v_{\max(i)}, \text{out})$.

The second optimization only concerns Algorithm 3, too. It is desirable to redistribute the excess outtoken flow of $v_{\max(i)}$ in each iteration step as uniform as possible among edges (v_j, v_l) with $j > \max(i)$ in order to produce as few as possible excess outtoken flows of such nodes v_j. In other words, $R(v_j, \chi_{i+1})$ should be as small as possible. This way, less nodes get under-saturated and thus also less nodes get over-saturated, and less exceed the outtoken flow overall (which must be redistributed in subsequent iteration steps) which is produced. There are several possibilities to implement this. First, it is possible to modify χ_i in two steps, first by saturating nodes v_j (this can be achieved by appropriate capacities) and then (if necessary) distributing remaining excess outtoken flow of $v_{\max(i)}$. The second possibility is to introduce costs for flow which over-saturates a node v_j and to compute the maximal flow with minimal costs (this can be done in polynomial time, too). The same idea can also be applied to the initial token flow function (up to now we have started with a big excess of the outtoken flow of the initial node v_0).

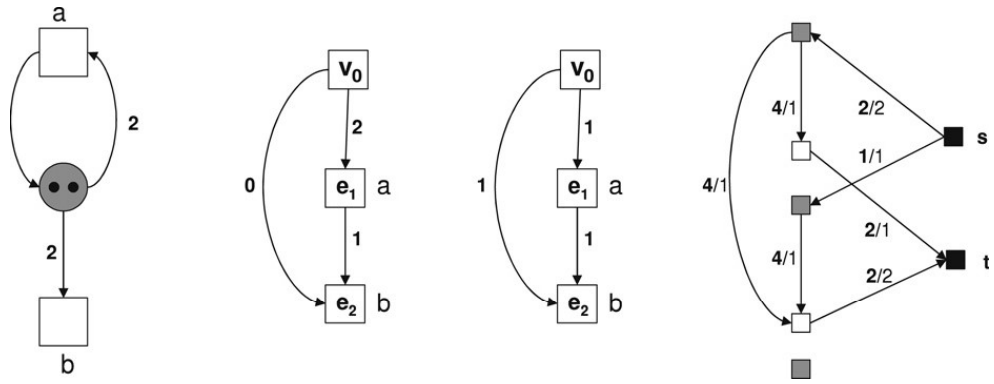


Fig. 8. From left to right: A marked p/t-net, an LPO with token flow function computed by Algorithm 3, the same LPO with token flow function computed by Algorithm 4, the associated flow network used by Algorithm 4 with maximal flow.

The last optimization applies to both Algorithm 3 and Algorithm 4. In general, there are edges (v, v') of lpo which only allow token flow 0 w.r.t. some place p, since this edge does not structurally appear in the p/t-net, that means $p \notin l(v)^\bullet \cap {}^\bullet l(v')$ for $v \neq v_0$ and $p \notin {}^\bullet l(v') \cap \{p \mid m_0(p) > 0\}$ for $v = v_0$. Such edges, of course, can be omitted in the construction of the flow network associated with lpo and p in both algorithms.

2.5. Comparing the algorithms

In this subsection, we compare the two Algorithms 3 and 4 w.r.t. their time complexity and w.r.t. the information they return in case an LPO is not an execution, in order to allow fault analysis.

Algorithm 4 returns less information about LPOs, which are not executions, than Algorithm 3. To illustrate this, let lpo not be an execution of (N, m_0) and let p be a place such that lpo does not fulfill the TFP w.r.t. p. Then Algorithm 3 (applied to p) terminates for some i after the i-th iteration because $\max(i) = \max(i-1)$. As described, from χ_i we are able to construct the set $D_f(\chi_i, p)$ defining a prefix of lpo and the cut $C(\chi_i, p)$ of lpo. We showed that the prefix defined by $D_f(\chi_i, p)$ is enabled w.r.t. p. Moreover, $C(\chi_i, p)$ is a cut in lpo subsequent to this prefix, and $C(\chi_i, p)$ is not enabled w.r.t. p after firing the prefix. Thus, $C(\chi_i, p)$ can be interpreted as a "bottleneck" of the "resource" p (notice that the prefix is not uniquely determined — it depends on χ_i and on the chosen total ordering of the nodes of lpo).

Algorithm 4 computes a "possible" token flow function χ_p for each place p. By construction χ_p fulfills (**OUT**) w.r.t. each node, but not necessarily (**IN**). Since the computation of the maximal flow need not respect the order of the nodes given by lpo, it is possible that there are two nodes v, v' with $v \prec v'$, where χ_p satisfies (**IN**) w.r.t. v', but not w.r.t. v.

Thus, it is in general not possible to construct from χ_p an enabled prefix of lpo followed by a cut representing a "bottleneck" of the "resource" p. This is illustrated in Fig. 8. The left part shows a marked p/t-net, the middle part shows an LPO annotated by two different token flow functions, and the right part shows the flow network associated with the net, annotated by pairs of capacity and flow values for some maximal flow (used for Algorithm 4). The LPO is not an execution w.r.t. the gray place of the net. The maximal flow in the flow network corresponds to the right token flow function in the middle part. This token flow function does not define a maximal prefix which is an execution, since (**IN**) is satisfied w.r.t. the b-labeled node, but not w.r.t. the a-labeled node, while the a-labeled node precedes the b-labeled node in the LPO. On the other hand, the left token flow function in the middle part defines such a maximal prefix (consisting of the a-labeled node). Note that Algorithm 3 would compute this left token flow function.

In order to use Algorithm 4 for the computation of enabled maximal prefixes similar as those in Algorithm 3, there are, in principle, two possibilities to modify Algorithm 4 (both increasing the runtime by one order in the number of nodes of the LPO).

- It would be possible to test iteratively bigger and bigger prefixes for enabledness.
- It would be possible to force Algorithm 4 to consider the nodes in some order, respecting the LPO by using flow costs for nodes and computing maximal flows with minimal costs (this problem also has polynomial solutions [29]).

We discuss the time complexity of the presented algorithms w.r.t. the number of edges e of the LPO, the number of nodes n of the LPO, the number of places q of the marked p/t-net and the maximal arc weight w of the marked p/t-net. We will compare the application of several maximal flow algorithms in the Algorithms 3 and 4. For both algorithms, the constructed flow networks have $O(e+n)$ edges and $O(n)$ nodes.

First, consider Algorithm 3. The maximal flow f_i in some iteration step i is bounded above by $R(v_{\max(i)}, \chi_i) = \mathrm{Out}(v_{\max(i)}, \chi_i) - W((l(v_{\max(i)}), p))$. The node $v_{\max(i)}$ has maximally $n - \max(i)$ successor nodes. Moreover, according to (**IN**), $\chi_i((v, v'))$ is bounded above by $W((p, l(v')))$ for each edge (v, v'). Therefore, $f_i \leqslant (n - \max(i)) \cdot w - W((l(v_{\max(i)}), p))) \leqslant n \cdot w$. The same applies to the maximal capacity value c_i of an edge in the flow network. In other words, f_i and c_i linearly depend on n and w. The chosen maximal flow algorithm is applied for each place, at most, n times (in the worst case, $\max(i)$ increases by one in each iteration step). The construction of the flow network in each iteration step and the computation of χ_{i+1}

take, at most, $O(e)$ time steps. Thus the maximal flow algorithm dominates. We deduce the following time complexities of Algorithm 3, applying different maximal flow algorithms: (i) $O(qwen^2)$ [28], (ii) $O(qn^4)$ [26], (iii) $O(qen^2 \log(n^2/e))$ [26] and (iv) $O(qen^2 + qn^3(\log wn)^{1/2})$ [27]. For LPOs with "few" edges ($e \leqslant O(n)$) and small w (compared to n) version (i) is most efficient. In particular, this is the case if w can be considered as a constant in applications. For flow networks with "many" edges ($e = O(n^2)$) the versions (ii)–(iv) are more efficient. If $O(n) < e < O(n^2)$, in most cases version (iv) is most efficient. Overall, which version is most efficient depends on the relationship of e to n.

Consider now Algorithm 4. The maximal flow f_i in the associated flow network is bounded from above by $M(\text{lpo}, p) \leqslant w \cdot n$. The same holds for the value c_i of the maximal capacity of an edge. Thus, an analogous argumentation as before yields the following time complexities, applying different maximal flow algorithms: (i) $O(qwen)$ [28], (ii) $O(qn^3)$ [26], (iii) $O(qen \log(n^2/e))$ [26] and (iv) $O(qen + qn^2(\log wn)^{1/2})$ [27].

2.6. Variants of executions

In this subsection, we briefly discuss other variants of executions. Instead of asking whether a given LPO *sequentializes a run* (i.e. whether it is an execution), we could also ask whether this LPO *equals a run*. Such LPOs we call *strict executions*. We could even be more restrictive and ask whether the LPO *equals a minimal run*. Such LPOs are called *minimal executions*. Finally, it is possible to consider the reverse direction and ask whether a given LPO *is sequentialized by a run*. This problem is a generalization of the so-called *legal firing sequence problem* (where one asks whether a given multi-set of transitions can be ordered to an enabled firing sequence), which was proven to be NP-hard ([30]).

2.6.1. Minimal executions

For the test of minimal executions, we presented the following polynomial algorithm (see [1]). Applying one of the Algorithm 3 or 4 yields one of the following three results:

- $\text{lpo} = (V, \prec, l)$ is not an execution. In this case lpo is not a minimal execution, too.
- lpo is an execution and for the run $(V, <, l)$ defined by the computed token flow functions it holds $<\subsetneq\prec$. In this case, lpo is not a minimal execution.
- lpo is an execution and for the run $(V, <, l)$ defined by the computed token flow functions it holds $\prec = <$. In this case lpo could be a minimal execution, but there could also be another run $(V, <', l)$ with $<' \subsetneq \prec$.

Thus, it is enough to consider the last case. For this case there is a simple strategy to test whether there is a run $(V, <', l)$ with $<' \subsetneq \prec$, namely simply to test whether some LPO $(V, <', l)$ with $<' \subsetneq \prec$ is an execution. Indeed, it is not necessary to consider all such LPOs, but only those which differ from lpo w.r.t. one skeleton edge. Formally, these are LPOs of the form $\text{lpo}_x = (V, \prec_x, l)$, where x is a skeleton edge and $\prec_x = \prec \setminus \{x\}$. It is easy to verify that $\prec_x$ is again transitive and therefore lpo_x is indeed an LPO. Algorithm 5 shows the procedure to test minimal executions.

Algorithm 5 (*Tests Whether* lpo *is a Minimal Execution of* (N, m_0)).
Step 1: Test if lpo is an execution of (N, m_0).
Step 2: Repeat for each edge $x \in \prec$: Test if lpo_x is an execution of (N, m_0).
Step 3: Return *true* if, and only if, lpo is an execution and no lpo_x is an execution of (N, m_0).

In the case, lpo is a minimal execution (i.e. a minimal run), it computes canonical token flow functions. Clearly, this algorithm runs in polynomial time, since the loop is passed through, at most, e times.

Theorem 14. *Let* lpo *be an execution of* (N, m_0). *Then* lpo *is a minimal execution if, and only if,* $\text{lpo}_x = (V, \prec_x, l)$ *is not an execution of* (N, m_0) *for each* $x \in \prec$.

2.6.2. Strict executions

The test of strict executions is more problematic, since not all runs of a p/t-net are minimal. Thus, even if $\text{lpo} = (V, \prec, l)$ equals a run, the Algorithm 3 or 4 possibly compute token flow functions, which define a run $(V, <, l)$ with $<\subsetneq\prec$.

A similar problem is, when given an LPO and a marked p/t-net, to find a run of this p/t-net which sequentializes the given LPO. This problem is a generalization of the so-called *legal firing sequence problem* which has no efficient solution.

One possibility to test whether an LPO is a strict execution, would be to strengthen the TFP in some way and to find a polynomial test of this stronger property. Observe that if lpo is an execution and $(V, <, l)$ is the run defined by the computed token flow functions χ_p for each place p, then $<=\prec$ holds if, and only if, for each skeleton edge e there is a place p with $\chi_p(e) > 0$. That means, lpo is a strict execution if, and only if, there exists a family of token flow functions $X = \{\chi_p \mid p \in P\}$ such that χ_p satisfies the TFP w.r.t. p and for each skeleton edge e there is a place p with $\chi_p(e) > 0$ (for example, the family of canonical token flow functions of a run is such a family). Unfortunately, computing such a family of token flow functions by maximal flows through appropriate flow networks does not longer yield an efficient algorithm in general. The problem is the additional requirement $\sum_{p \in P} \chi_p(e) \geqslant 1$ for skeleton edges e. That means, in the associated flow network we must also consider capacity bounds for the sum of several flows, each of which additionally has individual capacity bounds. This gives a so-called *multicommodity maximal flow problem*, which was proven to be NP-hard in most variants [31]. On the

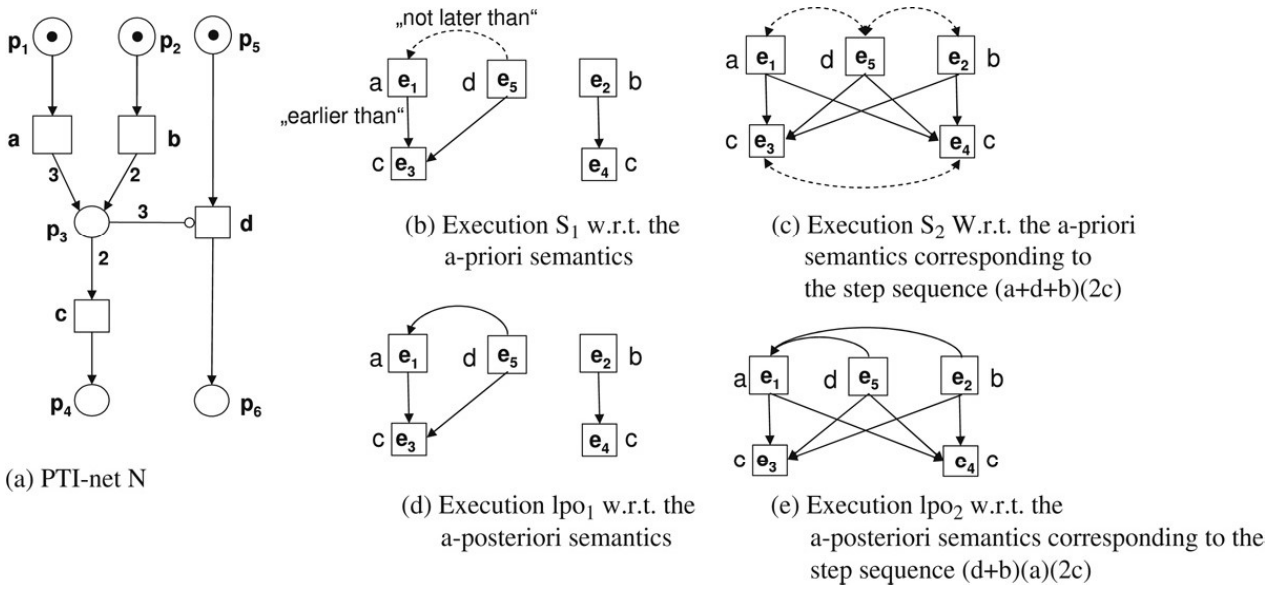


Fig. 9. A PTI-net with inhibitor arc (p_3, d) having weight 3 and executions of this net w.r.t. different semantics.

other hand, the instances we consider are restricted in some way compared to the most general version of multicommodity maximal flow problems. For example, all flows have the same source and the same sink. Moreover, there are no cycles in flow networks we consider (at least in the case of Algorithm 4). Whether these restrictions lead to polynomial algorithms is an open question.

3. PTI-nets

In this section we consider the problem of the executability of scenarios for PTI-nets, that means p/t-nets extended by weighted inhibitor arcs. Executions of such nets are given by more complex causal structures than LPOs, namely so-called stratified order structures. Their definition is based on relational structures. A *relational structure* (rel-structure) is a triple $\mathcal{S} = (V, \prec, \sqsubset)$, where V is a set (of *events*), and $\prec \subseteq V \times V$ and $\sqsubset \subseteq V \times V$ are binary relations on V. A rel-structure $\mathcal{S}' = (V, \prec', \sqsubset')$ is an *extension* of another rel-structure $\mathcal{S} = (V, \prec, \sqsubset)$, written $\mathcal{S} \subseteq \mathcal{S}'$, if $\prec \subseteq \prec'$ and $\sqsubset \subseteq \sqsubset'$.

A rel-structure $\mathcal{S} = (V, \prec, \sqsubset)$ is called *stratified order structure* (so-structure), if the following conditions are satisfied for all $u, v, w \in V$:(C1) $u \not\sqsubset u$, (C2) $u \prec v \implies u \sqsubset v$, (C3) $u \sqsubset v \sqsubset w \wedge u \neq w \implies u \sqsubset w$ and (C4) $u \sqsubset v \prec w \vee u \prec v \sqsubset w \implies u \prec w$. In figures, $\prec$ is graphically expressed by solid arcs and $\sqsubset$ by dashed arcs. According to (C2), a dashed arc is omitted, if there is already a solid arc. Moreover, we omit arcs, which can be deduced by (C3) and (C4) (see Fig. 9 (b), (c)).

It is shown in [20], that $(V, \prec)$ is a partial order. Therefore, so-structures are a generalization of partial orders and describe finer causalities than partial orders. In the context of this paper, $\prec$ represents an "earlier than"-relation, while $\sqsubset$ models a "not later than"-relation between events.

A so-structure $\mathcal{S} = (V, \prec, \sqsubset)$ is called *total linear* if $co_{\prec} = (\sqsubset \setminus \prec) \cup id_V$. The set of all *total linear extensions* (or *linearizations*) of a so-structure $\mathcal{S}$ is denoted by $lin(\mathcal{S})$ (see Fig. 9(c)).

A subset $W \subseteq V$ is called $\sqsubset$-*closed*, if $\forall v, v' \in V : (v \in W \wedge v' \sqsubset v) \implies v' \in W$. For $W \subseteq V$ $\sqsubset$-closed the so-structure $\mathcal{S}_W = (W, \prec|_{W\times W}, \sqsubset|_{W\times W})$ is called *prefix* of $\mathcal{S}$ defined by W. If additionally $(u \prec v \implies u \in W)$ for some $v \in V \setminus W$, then $\mathcal{S}_W$ is called *prefix of* $\mathcal{S}$ *enabling* v.

A *labeled so-structure* (LSO) is a so-structure $\mathcal{S} = (V, \prec, \sqsubset)$ together with a *set of labels* T and a *labeling function* $l : V \to T$. We also use the notations defined for so-structures for LSOs. As for LPOs, for $l : V \to T$ and $U \subseteq V$ we define the multi-set $l(U) \subseteq \mathbb{N}^T$ by $l(U)(t) = |\{v \in U \mid l(v) = t\}|$.

A *PTI-net* N is a quadruple (P, T, F, W, I), where (P, T, F, W) is a p/t-net, and $I : P \times T \to \mathbb{N} \cup \{\omega\}$ is the *weighted inhibitor relation*. If $I(p, t) \neq \omega$, then $(p, t) \in P \times T$ is called *(weighted) inhibitor arc*, and p is an *inhibitor place of* t. We define $n < \omega$ for $n \in \mathbb{N}$. A *marking* of a PTI-net $N = (P, T, F, W, I)$ is a function $m : P \to \mathbb{N}$. A *marked PTI-net* is a pair (N, m_0), where N is a PTI-net, and m_0 is a marking of N, called *initial marking*. Fig. 9(a) shows a marked PTI-net.

A transition t can be executed, if in addition to the enabling conditions of p/t-nets, every inhibitor place p of t carries at most $I((p, t))$ tokens. In particular, if $I((p, t)) = 0$, then p must be empty. $I((p, t)) = \omega$ means that t can never be prevented from occurring by the presence of tokens in p. There are two different semantics of PTI-nets concerning the order of the test of inhibitor restrictions and the production and consumption of tokens.

According to the *a priori semantics* of PTI-nets, the inhibitor test for enabledness of a transition precedes the consumption and production of tokens in places. Thus, a multi-set (a step) of transitions τ is (synchronously) enabled to occur in a marking m w.r.t. the a priori semantics, if $m(p) \geq \sum_{t\in T} \tau(t)W((p, t))$ and $m(p) \leq I((p, t))$ for each place p and transition $t \in \tau$.

According to the *a posteriori semantics* of PTI-nets, the inhibitor test for enabledness of a transition need not precede the consumption and production of tokens in places. It is even possible that the production of tokens precedes the consumption and the inhibitor test. Thus, a multi-set of transitions τ is enabled to occur in a marking m w.r.t. the a posteriori semantics, if $m(p) \geq \sum_{t\in T} \tau(t)W((p,t))$ and $m(p) + \sum_{t\in T} \tau(t)W((t,p)) \leq I((p,t))$ for each place p and transition $t \in \tau$.

The *occurrence* of a (possibly empty) step of transitions τ (in the a priori or a posteriori semantics) leads to the new marking m', defined by $m'(p) = m(p) - \sum_{t\in T} \tau(t)(W((p,t)) - W((t,p)))$ for every $p \in P$. We write $m \xrightarrow{\tau} m'$ to express, that τ is enabled to occur in m, and that its occurrence leads to m'. A finite sequence of steps $\sigma = \tau_1 \dots \tau_n,\ n \in \mathbb{N}$, is called a *step occurrence sequence enabled in a marking m and leading to* m_n, if there exists a sequence of markings $m_1, \dots, m_n$ such that $m \xrightarrow{\tau_1} m_1 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_n} m_n$. In this case we write $m \xrightarrow{\sigma} m_n$

3.1. Causal semantics

Up to now, there is no unique acknowledged process semantics of nets with inhibitor arcs w.r.t. a priori- or a posteriori-semantics, but only several proposals [20,21,14,32,12,33]. We omit to present these process semantics here, and base the definition of causal semantics on step semantics (see also the Introduction). That means, in this subsection we lift the notions of *enabled LPO* and *token flow property (TFP)*, known for LPOs w.r.t. p/t-nets, to the setting of PTI-nets.

For the a posteriori semantics, executions of PTI-nets are given by LPOs. That means causal semantics can be given, as in the case of p/t-nets, by identifying step occurrence sequences with LPOs (Fig. 9(e)). We call an LPO $\mathrm{lpo} = (V, \prec, l)$ *enabled (to occur) w.r.t. a marked PTI-net in the a posteriori semantics* if each finite step sequence $\sigma = \tau_1 \dots \tau_n$ which sequentializes lpo is a step occurrence sequence of the PTI-net in the a posteriori semantics (Fig. 9(d) and (e)). We say that the occurrence of lpo *leads* to the marking $m'(p)$ given by $m'(p) = m(p) + \sum_{v\in V}(W((l(v),p)) - W((p,l(v))))$.

For the a priori semantics, executions of PTI-nets are given by LSOs. The notion of enabled LPOs can be straightforwardly extended to enabled LSOs using step occurrence sequences. As in the LPO-case, a step sequence of transitions $\sigma = \tau_1 \dots \tau_n$ can be identified with the LSO $\mathcal{S}_\sigma = (V, \prec, \sqsubset, l)$ defined by $V = \bigcup_{i=1}^{n} V_i$ and $l : V \to T$ with $l(V_i) = \tau_i$, $\prec = \bigcup_{i<j} V_i \times V_j$ and $\sqsubset = ((\bigcup_i V_i \times V_i) \cup \prec) \setminus id_V$ (Fig. 9(c)). Such LSOs are total linear (because $co_\prec = \bigcup_{i=1}^{n} V_i \times V_i$). The other way round, each total linear LSO (of transition occurrences) can be identified with a step sequence of transitions. Therefore, we call an LSO $\mathcal{S} = (V, \prec, \sqsubset, l)$ with $l : V \to T$ *enabled (to occur) w.r.t. a marked PTI-net in the a priori semantics*, if each finite step sequence $\sigma = \tau_1 \dots \tau_n$ with $\mathcal{S}_\sigma \in lin(\mathcal{S})$ is a step occurrence sequence of the PTI-net. We say that the occurrence of $\mathcal{S}$ *leads* to the marking $m'(p)$, given by $m'(p) = m(p) + \sum_{v\in V}(W((l(v),p)) - W((p,l(v))))$. It is easy to check, that the LSOs from Fig. 9(b) and (c) are indeed enabled LSOs w.r.t. the shown PTI-net.

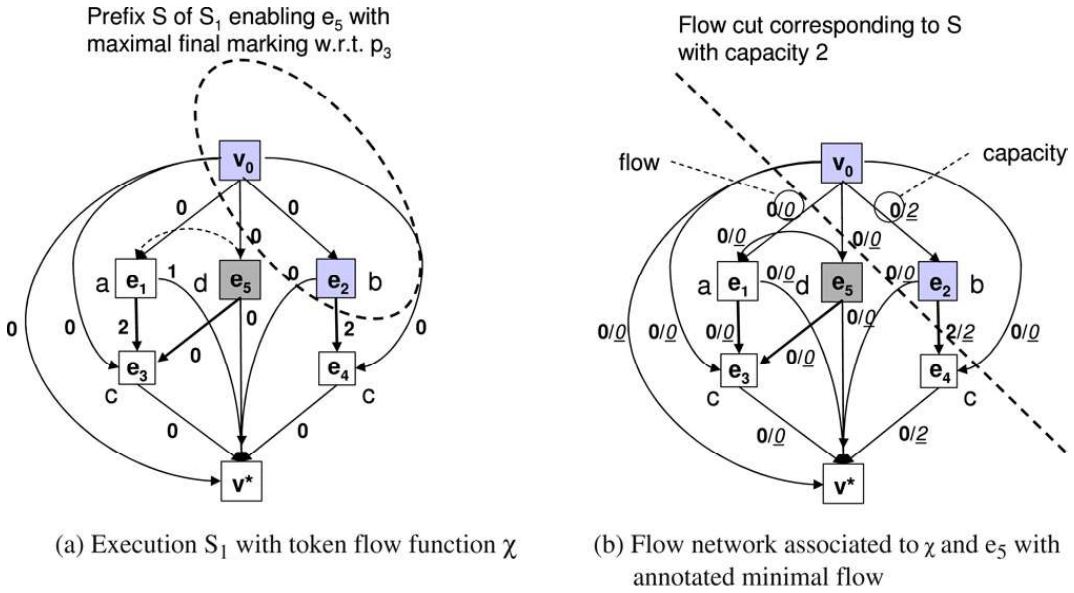
3.2. A priori semantics

For the development of the TFP and the polynomial test of the TFP w.r.t. PTI-nets, we first consider their a priori semantics. The content of this subsection was presented in [2].

3.2.1. Token flow property

In this subsection, we extend the notions of token flow function and TFP, known for LPOs and p/t-nets, to the setting of PTI-nets w.r.t. the a priori semantics. Fix a marked PTI-net (N, m_0), $N = (P, T, F, W, I)$, a place p of N and an LSO $\mathcal{S} = (V, \prec, \sqsubset, l)$ with $l : V \to T$. Assume, that $\mathcal{S}$ is enabled to occur w.r.t. (N, m_0). Since the inhibitor relation I of (N, m_0) restricts the behavior of the *underlying p/t-net* $(N', m_0) = (P, T, F, W, m_0)$, $\mathcal{S}$ is also enabled w.r.t. (N', m_0). In a p/t-net, transitions which can be executed as one step also can be executed in arbitrary order. Therefore, the LPO $\mathrm{lpo}_\mathcal{S} = (V, \prec, l)$ *underlying* $\mathcal{S}$ is also enabled w.r.t. the p/t-net (N', m_0). Altogether, we get that the enabledness of $\mathrm{lpo}_\mathcal{S}$ w.r.t. the p/t-net (N', m_0) is a necessary condition for the enabledness of $\mathcal{S}$ w.r.t. (N, m_0). That means, the TFP for $\mathcal{S}$ w.r.t. (N, m_0) includes the TFP for $\mathrm{lpo}_\mathcal{S}$ w.r.t. (N', m_0). Since the "not later than"-relation of $\mathcal{S}$ does not describe the flow of tokens (token flow always produces an "earlier than"-relation between transition occurrences), a token flow function of $\mathcal{S}$ w.r.t. a place can be given by a token flow function of $\mathrm{lpo}_\mathcal{S}$. As argued above, if $\mathcal{S}$ is enabled, then for each place p there is such a token flow function χ_p satisfying (**IN**) and (**OUT**). The other way round the existence of such token flow functions is not enough to ensure that $\mathcal{S}$ is enabled. This is because the execution of a prefix of $\mathcal{S}$ might still produce too many tokens in a place p (according to χ_p), disabling a subsequent transition, which tests p via an inhibitor arc. In other words, the maximal number of tokens (according to χ_p) produced in p after the occurrence of a prefix should not exceed the inhibitor weights. To ensure this, we require that token flow functions fulfill an additional property. This property implies that each marking enabling some event, which is reachable through the execution of a prefix, respects the inhibitor relations of the corresponding transition to all places.

In order to efficiently compute the maximal number of tokens (according to χ_p) produced in p after the occurrence of a prefix, it is convenient to use slightly different notions of 0-extensions of LPOs, token flow functions and the TFP for LPOs. The new notion of 0-extensions also adds a new maximal event $v^\star$, which is interpreted as an event consuming the final marking reached after the occurrence of an LPO, to LPOs. Token flow functions are then defined on these new 0-extensions, leading to a slightly different but equivalent notion of the TFP. Namely, we now require that the outtoken flow of each node *equals* the corresponding arc weight in the net. Then the old concept of token flow functions can be translated into the new one (and vice versa) via the identification $\chi((v, v^\star)) = R(v, \chi)$.



(a) Execution S_1 with token flow function χ

(b) Flow network associated to χ and e_5 with annotated minimal flow

Fig. 10. (New) 0-extension of the LSO S_1 from Fig. 9 (b) with associated flow network. S_1 is enabled w.r.t. the PTI-net N from Fig. 9 (a).

Definition 15 (*Equivalent TFP*). Let $\mathrm{lpo} = (V, \prec, l)$ be an LPO. Then an LPO $\mathrm{lpo}^0 = (V^0, \prec^0, l^0)$, where $V^0 = (V \cup \{v_0, v^\star\})$, $v_0, v^\star \notin V$, $\prec^0 = \prec \cup (\{v_0\} \times V) \cup ((V \cup \{v^0\}) \times \{v^\star\})$, and $l^0(v_0) \neq l^0(v^\star),\ l^0(v_0), l^0(v^\star) \notin l(V), l^0|_V = l$, is called 0-*extension of* lpo.

A function $\chi : \prec^0 \to \mathbb{N}$ is called *token flow function* of lpo, if it satisfies (**Tff**) $\forall v, v' \in V^0 :\ l(v) = l(v') \implies \mathrm{In}(v, \chi) = \mathrm{In}(v', \chi)$.

Denote $W((l(v_0), p)) = m_0(p)$ for each place $p \in P$. Then lpo fulfills the *token flow property (TFP) w.r.t.* (N, m_0) if, for all $p \in P$, there is a token flow function $\chi_p : \prec^0 \to \mathbb{N}$ satisfying (**IN**) $\forall v \in V :\ \mathrm{In}(v, \chi_p) = W((p, l(v)))$ and (**OUT**) $\forall v' \in V \cup \{v^0\} :\ \mathrm{Out}(v', \chi_p) = W((l(v'), p))$.

Assume, that we have given a (new) token flow function χ_p on the edges of $\mathrm{lpo}^0_{\mathcal{S}}$, satisfying (**IN**) and (**OUT**) for some place p. How can we compute from χ_p the number of tokens in this place after the execution of some prefix of $\mathcal{S}$? Let V' define a prefix. The values of χ_p on edges between events in V' correspond to tokens, which are produced and consumed by events in this prefix. The values of χ_p on edges from events in V' to events in $V \setminus V'$ correspond to tokens, which are produced by events in V' and remain in p after the execution of the prefix. Thus, the marking of the place after the execution of the prefix is given by the sum of the values of χ_p on such edges (Fig. 10(a)).

Definition 16 (*Final Marking*). Let $\mathcal{S}' = (V', \prec', \sqsubset', l')$ be a prefix of $\mathcal{S}$ and $\chi : V^0 \to \mathbb{N}$ be a token flow function of $(V, \prec, l)$. The *final marking* $m_{\mathcal{S}'}(\chi)$ *of* $\mathcal{S}'$ *(w.r.t.* χ*)* is defined by $m_{\mathcal{S}'}(\chi) = \sum_{u \in V' \cup \{v_0\},\ v \notin V',\ u \prec^0 v} \chi((u, v))$.

If a token flow function fulfills (**IN**) and (**OUT**) then the final marking of a prefix in fact does not depend on the concrete distribution of the token flow given by this token flow function, but only on the nodes belonging to the prefix. In this case, the final marking can be computed (independently from the token flow function) also by $m_{\mathcal{S}'}(\chi) = m_0(p) + \sum_{t \in T} l(V')(t)(W((t, p)) - W((p, t)))$.

Definition 17 (*Token Flow Property*). An LSO $\mathcal{S} = (V, \prec, \sqsubset, l)$ fulfills the *token flow property w.r.t.* (N, m_0), if for every place $p \in P$ there exists a token flow function $\chi_p : \prec^0 \to \mathbb{N}$, satisfying (**IN**), (**OUT**) and
(**FIN**) For all $v \in V$ and all prefixes $\mathcal{S}'$ enabling v: $m_{\mathcal{S}'}(\chi_p) \leqslant I((p, l(v)))$.

Observe that the definition of the TFP is an inherent exponential in the size of the LSO, since it involves, in general, exponentially many prefixes of the LSO (condition (**FIN**)). Nonetheless, as will be explained in Section 3.2.2, the test of condition (**FIN**) can be transformed into a flow optimization problem, which can be solved in polynomial time. The following lemma and theorem show that the TFP is an equivalent notion of executions.

Lemma 18. *Let* $\mathcal{S} = (V, \prec, \sqsubset)$ *be a so-structure,* $V' \subseteq V$ *and* $v \in V \setminus V'$*. Then,* V' *defines a prefix of* $\mathcal{S}$ *enabling* v*, if, and only if, there is a linearization* $\mathcal{S}' \in \mathit{lin}(\mathcal{S})$*, such that* V' *defines a prefix of* $\mathcal{S}'$ *enabling* v*.*

Proof. if: Let $\mathcal{S}' = (V, \prec', \sqsubset') \in \mathit{lin}(\mathcal{S})$ and let $V' \subset V$ define a prefix of $\mathcal{S}'$ enabling v. Consider nodes $u' \in V'$ and $u \in V$ with $u \sqsubset u'$. Since $\mathcal{S}'$ is an extension of $\mathcal{S}$, this implies $u \sqsubset' u'$. Because V' defines a prefix of $\mathcal{S}'$, we get $u \in V'$. Thus, V' also defines a prefix of $\mathcal{S}$. Further, let $v' \prec v$. Again, since $\mathcal{S}'$ is an extension of $\mathcal{S}$, this implies $v' \prec' v$, and therefore we have $v' \in V'$. Thus, V' defines, in fact, a prefix enabling v.

only if: Let V' define a prefix of $\mathcal{S}$ enabling v. We construct a linearization $\mathcal{S}' = (V, \prec', \sqsubset')$ of $\mathcal{S}$, such that V' also defines a prefix of $\mathcal{S}'$ enabling v. For this, let $V_0 \subseteq V'$ be the set of all nodes, which are minimal w.r.t. $\prec$ in $\mathcal{S}$. Then, consider the

restriction of $\mathcal{S}$ onto the node set $V \setminus V_0$ and let $V_1 \subseteq V'$ be the set of all nodes, which are minimal w.r.t. $\prec$ in this new so-structure. Following this technique, we define inductively $V_n \subseteq V'$ as the set of nodes, which are minimal w.r.t. the restriction of $\prec$ onto the node set $V \setminus (\bigcup_{i=0}^{n-1} V_i)$, as long as $V' \setminus (\bigcup_{i=0}^{n-1} V_i) \neq \emptyset$. Let N be minimal with the property $V' \setminus (\bigcup_{i=0}^{N-1} V_i) = \emptyset$. We further define $V_N \subseteq V$ as the set of nodes, which are minimal w.r.t. the restriction of $\prec$ onto the node set $V \setminus (\bigcup_{i=0}^{N-1} V_i)$, and so on (note that $v \in V_N$, because V' defines a prefix enabling v).

We now can define $\mathcal{S}'$ through $\prec' = \bigcup_{i<j} V_i \times V_j$ and $\sqsubset' = ((\bigcup_i V_i \times V_i) \setminus id_{V_i}) \cup \prec'$. By construction, $\mathcal{S}'$ is a total linear so-structure. It remains to show that $\prec \subseteq \prec'$, $\sqsubset \subseteq \sqsubset'$, $((u \in V' \wedge w \sqsubset' u) \implies w \in V')$ and $(v' \prec' v \implies v' \in V')$.

Let $u, v \in V$ with $u \prec v$: Since V' defines a prefix of $\mathcal{S}$, it is not possible that $v \in V'$ and $u \notin V'$. Suppose $u, v \in V'$, $u, v \in V \setminus V'$ or $u \in V'$ and $v \notin V'$. By construction there must be $i < j$ with $u \in V_i$ and $v \in V_j$. This gives $u \prec' v$.

Let $u, v \in V$ with $u \sqsubset v$: Since V' defines a prefix of $\mathcal{S}$, it is not possible that $v \in V'$ and $u \notin V'$. Suppose $u, v \in V'$ or $u, v \in V \setminus V'$: Let $u \in V_i$ and $v \in V_j$. Assume, that v is minimal w.r.t. $\prec$ in an earlier step than u. Then in this step, there holds $u' \prec u$ but $u' \not\prec v$. This contradicts (C4). Therefore either u and v are minimal in the same step or u is minimal in a step earlier than v. This gives $u \sqsubset' v$. Suppose $u \in V'$ and $v \notin V'$: Then by construction, there must be $i < j$ with $u \in V_i$ and $v \in V_j$. This gives $u \prec' v$.

Let $u \in V'$ and $w \sqsubset' u$: Then by construction $w \in V_i$ and $u \in V_j$ for some $i < j < N$ or $u, w \in V_i$ for some $i < N$. This implies $w \in V'$.

Let $v' \in V$ with $v' \prec' v$: Since by construction $v \in V_N$, there is $i < N$ with $v' \in V_i \subseteq V'$.

Theorem 19. *$\mathcal{S}$ is enabled w.r.t. (N, m_0) (a priori semantics) if, and only if, it fulfills the TFP w.r.t. (N, m_0).*

Proof. only if: Let $\mathcal{S}$ be enabled w.r.t. (N, m_0). Then $(V, \prec, l)$ is enabled w.r.t. (P, T, F, W, m_0), that means for each $p \in P$, there is a token flow function $\chi_p : \prec^0 \to \mathbb{N}$ of $(V, \prec, l)$, satisfying (**IN**) and (**OUT**). We claim, that each χ_p also fulfills (**FIN**).

Let $v \in V$ and $\mathcal{S}'$ be a prefix of $\mathcal{S}$ defined by V' which enables v'. By Lemma 18, there is a linearization $\mathcal{S}_{lin}$ of $\mathcal{S}$, such that V' defines a prefix $\mathcal{S}'_{lin}$ of $\mathcal{S}_{lin}$ which enables v. There is a step occurrence sequence $\sigma = \tau_1 \dots \tau_n$ of (N, m_0) with $\mathcal{S}_\sigma = \mathcal{S}_{lin}$. Since prefixes are downward $\sqsubset$-closed, a prefix $\sigma' = \tau_1 \dots \tau_m$ $(m < n)$ of σ with $l(v) \in \tau_{m+1}$ and $\mathcal{S}_{\sigma'} = \mathcal{S}'_{lin}$ (up to isomorphism) must exist. The statement follows from $m'(p) = m_{\mathcal{S}'}(\chi_p)$ for the marking m' reached after the execution of σ', since $m'(p) \leq I((p, t))$ for each place p and each transition $t \in \tau_{m+1}$ by the definition of step occurrence sequences.

if: Let $\mathcal{S}$ fulfill the TFP w.r.t. (N, m_0), and let χ_p be a token flow function satisfying (**IN**), (**OUT**) and (**FIN**) w.r.t. the place p. Consider a sequence of transition steps $\sigma = \tau_1 \dots \tau_n$ such that $\mathcal{S}_\sigma$ is a linearization of $\mathcal{S}$. We show inductively that, if $\sigma_k = \tau_1 \dots \tau_k$ is a step occurrence sequence, then τ_{k+1} is a transition step, enabled in the marking m' reached after the execution of σ_k for $0 \leqslant k \leqslant n-1$.

Observe that σ is a step occurrence sequence of the p/t-net (P, T, F, W, m_0), since $(V, \prec, l)$ satisfies the token flow property on the p/t-net level and σ sequentializes $(V, \prec, l)$. That means, $m'(p) \geq \sum_{t \in \tau_{k+1}} \tau_{k+1}(t) W((p, t))$ is always satisfied. It remains to verify that $m'(p) \leq I((p, t))$ for each place p and each transition $t \in \tau_{k+1}$. We have that $\mathcal{S}_{\sigma_k} = (V_k, \prec_k, \sqsubset_k, l_k)$ is a prefix of $\mathcal{S}_\sigma$. By Lemma 18, V_k also defines a prefix $\mathcal{S}_k$ of $\mathcal{S}$. Fix $t \in \tau_{k+1}$ and $p \in P$ and let $v \in V$ with $l(v) = t$, such that $\mathcal{S}_{\sigma_k}$ is a prefix which enables v. Then, also $\mathcal{S}_k$ is a prefix which enables v (Lemma 18). As above, the statement follows from $m'(p) = m_{\mathcal{S}_k}(\chi_p)$, since $m_{\mathcal{S}_k}(\chi_p) \leqslant I((p, l(v))$ by (**FIN**).

3.2.2. Polynomial test

In this section, we give a polynomial algorithm to test whether an LSO $\mathcal{S} = (V, \prec, \sqsubset, l)$ with $l(V) = T$ fulfills the TFP w.r.t. a marked PTI-net (N, m_0). In the case that $\mathcal{S}$ fulfills the TFP, the algorithm constructs respective token flow functions for every place, satisfying (**IN**), (**OUT**) and (**FIN**).

Algorithm 3 tests in polynomial time, whether for each place there is a token flow function satisfying (**IN**) and (**OUT**). If such token flow functions do not exist, then the LSO does not fulfill the TFP. In the positive case, Algorithm 3 generates such token flow functions. Either these token flow functions satisfy (**FIN**), or the LSO does not fulfill the TFP, since the final marking of a prefix w.r.t. a place p only depends on the initial marking $m_0(p)$ and the arc weights $W((p, t))$ and $W((t, p))$ for $t \in T$, but not on the concrete distribution of the token flow. That means, for different token flow functions χ_p and χ'_p, satisfying (**IN**) and (**OUT**) for a place p, the values $m_{\mathcal{S}'}(\chi_p)$ and $m_{\mathcal{S}'}(\chi'_p)$ coincide. Thus, either χ_p and χ'_p both fulfill (**FIN**), or both do not fulfill (**FIN**). It remains to test property (**FIN**) for the computed token flow functions χ_p satisfying (**IN**) and (**OUT**). For this, it is enough to compute for each node v the maximum of the values $m_{\mathcal{S}'}(\chi_p)$ over all prefixes $\mathcal{S}'$ enabling v and to compare this maximum with the value $I((p, l(v)))$.

Definition 20 (*Inhibitor Value*). The *inhibitor value* $Inh(v, \chi)$ of an event v w.r.t. a token flow function χ is defined by $Inh(v, \chi) = \max\{m_{\mathcal{S}'}(\chi) \mid \mathcal{S}' \text{ is a prefix enabling } v\}$.

A straightforward way to compute the inhibitor value of some node v is to enumerate all prefixes enabling this node and compute the final markings of all these prefixes. Unfortunately, this is not efficient, since there may be exponentially many prefixes in the number of nodes. Another possible formalization of the problem is as follows. The final marking of a prefix is defined as the sum over the values of the token flow function on edges leaving the prefix. These edges separate the node set of the prefix from the subsequent nodes. Formally, this separation is a flow cut through $\mathcal{S}$ (resp. $\mathrm{lpo}_\mathcal{S}$), partitioning the set of nodes of $\mathcal{S}$ into two node sets. Interpreting $\mathrm{lpo}_\mathcal{S}$ as a flow network and the values of the token flow function as lower

capacity bounds for flows through this network, the final marking of a prefix is given as the capacity of some flow cut, and the inhibitor value of some node can be seen as the maximum capacity of flow cuts of the network.

Such a maximum capacity can be efficiently computed through considering *flow networks with lower capacities* and *minimal flows* through such networks. This variant of flow optimization problems can be seen as the reversed maximal flow problem. It can be proven analogously that in such networks there is no *flow decreasing path* w.r.t. a flow f if, and only if, f is minimal and that the minimal flow equals the *maximal capacity of a cut*. Moreover, solution algorithms of the maximal flow problem based on flow augmenting paths can easily be adapted (for example the algorithm from [25]). This can be briefly seen as follows:

Fix a flow network (G, c, s, t), $G = (W, E)$ and consider *flows* f in (G, c, s, t) satisfying $\forall (v, v') \in E : f((v, v')) \geqslant c((v, v'))$. The capacity of a flow cut (S, T) in (G, c, s, t) is defined by $c((S, T)) = \sum_{v \in S,\, w \in T,\, (v,w) \in E} c((v, w))$ if $(T \times S) \cap E = \emptyset$ and $c((S, T)) = 0$ else. The *residual network* (G, c_f, s, t), $G = (W, E_f)$, w.r.t. a flow f is defined as follows: For $(v, v') \in E$ define $c_f((v, v')) = f((v, v')) - c((v, v'))$ and set $E_f = \{(v, v') \in W \times W \mid ((v, v') \in E \wedge c_f((v, v')) > 0) \vee ((v', v) \in E)\}$. A *flow reducing path w.r.t. a flow* f in the residual network is a simple path from source to sink of the residual network. Then it holds:

Theorem 21. *The following statements are equivalent:* (i) *f is a minimal flow,* (ii) *There is no flow reducing path in the* residual network w.r.t. *f,* (iii) *There is a flow cut (S, T) with $(T \times S) \cap E = \emptyset$ and $c((S, T)) = |f|$.*

Proof. (i) $\Longrightarrow$ (ii): Let f be a minimal flow and assume there is a flow reducing path in the residual network. Then, along this flow reducing path the flow f can be reduced. This contradicts the minimality of f. The reduction is as follows. For edges $(v, v') \in E$, if (v, v') belongs to the path then reduce the flow on this edge by 1, if (v', v) belongs to the path then augment the flow on this edge by 1. Then, by construction, the modified flow still satisfies the capacity constraint. Also the second defining property of flows, saying that the flow ingoing to a node equals the flow outgoing from a node, is still satisfied. Either the flow ingoing to (outgoing from) a node is once reduced and once augmented by 1 (along the path) or ingoing and outgoing flow of a node are both reduced or both augmented by 1 (along the path). Moreover $|f|$ is reduced by 1 since this is the case for the flow ingoing to the sink.

(ii) $\Longrightarrow$ (iii): Assume there is no flow reducing path in the residual network w.r.t. f. We define a flow cut (S, T) as follows: $S = \{w \in W \mid \text{there is a simple path from } s \text{ to } w \text{ in the residual network w.r.t. } f\}$ and $T = W \setminus S$. It follows that $f((u, v)) = c((u, v))$ for each edge $(u, v) \in E \cap (S \times T)$, because otherwise $(u, v) \in E_f$, i.e. $v \in S$. Moreover, we deduce $E \cap (T \times S) = \emptyset$, because otherwise $(v, u) \in E_f$, i.e. $u \in S$, for each $(u, v) \in E \cap (T \times S)$. It is easy to see that $|f| = \sum_{e \in E \cap (S \times T)} f(e) - \sum_{e \in E \cap (T \times S)} f(e)$ (for each flow cut (S, T)). This gives $c((S, T)) = |f|$.

(iii) $\Longrightarrow$ (i): Finally, if there is a flow cut (S, T) with $|f| = c((S, T))$ then f must be minimal since $|f| = \sum_{e \in E \cap (S' \times T')} f(e) - \sum_{e \in E \cap (T' \times S')} f(e) \geqslant c((S', T'))$ for all flow cuts (S', T') (because $c((S', T')) = 0$ in the case $E \cap (T' \times S') \neq \emptyset$). In particular, it holds $|f| = c((S, T))$ if, and only if, (S, T) is a flow cut with maximal capacity and f is a minimal flow in the flow network.

To compute a minimal flow in a flow network with lower capacities, first we compute an arbitrary (feasible) flow of the flow network satisfying the lower capacity constraint by a transformation into a maximal flow problem [24]. Then we can use, for example, an adaptation of the algorithm from [25] using flow reducing paths instead of flow augmenting paths, to reduce the flow step by step. This takes maximal $O(n^3)$ time.

Altogether, the maximum capacity can be computed efficiently through its correspondence to minimal flows. We now formally construct the flow network (Fig. 10(b)). For this, we interpret $\mathcal{S}$ as a flow network. We first omit the "not later than"-relation as follows. We can glue events of $\mathcal{S}$, which are in a symmetric "not later than"-relation. If $u \sqsubset v$ but $v \not\sqsubset u$, then there might be prefixes containing u but not v, and there might be prefixes, which contain or do not contain both events u and v together. Since the same holds if $u \prec v$, we replace remaining "not later"-than relations by "earlier than"-relations. We do not want to consider all flow cuts of this flow network, but only those, corresponding to prefixes enabling v. Therefore, we only define (lower) capacity constraints on edges leaving a prefix enabling v.

Definition 22 (*Associated Flow Network*). Let $v \in V$ and $\chi : \prec^0 \to \mathbb{N}$ be a token flow function of $\mathcal{S}$. Further, let U be the set of all the nodes occurring in prefixes enabling v, including v_0, and define $[u] = [u]_{\sqsubset} = \{w \in V^0 \mid w = u \vee (w \sqsubset^0 u \wedge u \sqsubset^0 w)\}$ for $u \in V^0$.

Define the *flow network* (G, c, s, t), $G = (W, E)$, *associated to* χ *and* v by $W = \{[u] \mid u \in V^0\}$, $s = [v_0]$ $(= \{v_0\})$, $t = [v^\star]$ $(= \{v^\star\})$, $E = \{([u], [w]) \mid u \sqsubset^0 w\}$ and $c(([u], [w])) = \sum_{u' \in [u],\, w' \in [w],\, u' \prec^0 w'} \chi((u', w'))$ if $u \in U \wedge w \not\prec v$ and $c(([u], [w])) = 0$ else.

Let V' define a prefix of $\mathcal{S}$. Then the flow cut $(S_{V'}, T_{V'})$ *corresponding to* V' is defined by $S_{V'} = \{[v] \mid v \in V' \cup \{v_0\}\}$ and $T_{V'} = W \setminus S_{V'}$.

Observe that the associated flow network is well-defined. That means for $u' \in [u]$ and $w' \in [w]$, we have $u \sqsubset^0 w \Longrightarrow u' \sqsubset^0 w'$ and $c(([u], [w])) = c(([u'], [w']))$. The following lemma states, that the final marking of prefixes enabling v can be computed by capacities of flow cuts in the associated flow network.

Lemma 23. *Let $\mathcal{S}' = (V', \prec', \sqsubset', l')$ be a prefix enabling an event v. Further, let χ be a token flow function of $\mathcal{S}$, and (G, c, s, t), $G = (W, E)$, be the flow network associated to χ and v. Then $m_{\mathcal{S}'}(\chi) = c((S_{V'}, T_{V'}))$.*

Proof. Since $\{w \mid w \prec v\} \subseteq V' \subseteq U$, we have for each $u \in V' \cup \{v_0\}$ and $w \notin V' \cup \{v_0\}$ that $c(([u],[w])) = \sum_{u' \in [u],\, w' \in [w],\, u' \prec^0 w'} \chi((u', w'))$. The statement is now an easy computation. Just observe, that $(T_{V'} \times S_{V'}) \cap E = \emptyset$ since $w \not\sqsubset^0 u$ for $[u] \in S_{V'}$, $[w] \in T_{V'}$.

Since flow cuts which do not correspond to prefixes enabling v do not have bigger capacities than flow cuts corresponding to such prefixes, we get:

Theorem 24. *Let v be a node, and $\chi : \prec^0 \to \mathbb{N}$ be a token flow function of $\mathrm{lpo}_{\mathcal{S}}$. Further, let (G, c, s, t), $G = (W, E)$, be the flow network associated to χ and v. Then $Inh(v, \chi) = \max\{c((S,T)) \mid (S,T)$ flow cut of $(G, c, s, t)\}$.*

Proof. Let (S, T) be a flow cut of (G, c, s, t) not corresponding to a prefix enabling v. We have to show that there is a flow cut corresponding to a prefix enabling v which has a bigger capacity. Then the statement follows from Lemma 23. There are two cases to distinguish.

First, let (S, T) not correspond to a prefix of $\mathcal{S} = (V, \prec, \sqsubset, l)$. In this case $V_S = \bigcup_{[u] \in S \setminus [v_0]} [u]$ does not define a prefix of $\mathcal{S}$. That means that there is $u \in V_S$ and $w \notin V_S$ with $w \sqsubset u$. By the definition of V_S, it is not possible that also $u \sqsubset w$ (because then $[w] = [u]$). Therefore, by the definition of E, we get $([w], [u]) \in E$. This implies $c((S, T)) = 0$.

Second, let $(S, T) = (S_{V'}, T_{V'})$ correspond to a prefix $\mathcal{S}' = (V', \prec', \sqsubset', l')$ not enabling v. We claim that, then, there is a prefix $\mathcal{S}_0 = (V_0, \prec_0, \sqsubset_0, l_0)$ enabling v such that $c((S_{V'}, T_{V'})) \leqslant c((S_{V_0}, T_{V_0}))$. Observe that the intersection and the union of two node sets, which both define prefixes (enabling v) always defines a prefix (which enables v) again. This implies that there is a maximal prefix which enables v (it equals U) and also a minimal prefix which enables v defined by the set $U' = \{u \in V \mid u \prec v\}$. In particular, the intersection $V'' = V' \cap U$ defines a prefix $\mathcal{S}''$. Then $c((S_{V'}, T_{V'})) \leqslant c((S_{V''}, T_{V''}))$, since $c(([u],[w])) = 0$ if $u \notin U$ (only such edges count for the flow cut $(S_{V'}, T_{V'})$ but not for the flow cut $(S_{V''}, T_{V''})$), and there may be edges $([u],[w]) \in E$ with $u \in V''$ and $w \in V' \setminus V''$ (these edges count for $(S_{V''}, T_{V''})$ but not for $(S_{V'}, T_{V'})$). Finally, $V_0 = V'' \cup U'$ defines a prefix enabling v with $c((S_{V''}, T_{V''})) \leqslant c((S_{V_0}, T_{V_0}))$. This follows, since $c(([u],[w])) = 0$ if $w \prec v$ (only such edges count for $(S_{V''}, T_{V''})$ but not for (S_{V_0}, T_{V_0})), and there may be edges $([u],[w]) \in E$ with $u \in V_0 \setminus V''$ (these edges count for (S_{V_0}, T_{V_0}) but not for $(S_{V''}, T_{V''})$).

Thus, inhibitor values can be computed through the maximal capacity of a flow cut in an appropriate flow network. This maximal capacity equals the minimal flow through this network. This minimal flow can be computed in polynomial time (an explanation of the main arguments can be found in the Appendix). If p is a place for which there is a token flow function satisfying (**IN**) and (**OUT**), then the inhibitor value w.r.t. this token flow function must be computed for each node of the LSO. A comparison of these inhibitor values and the weights of the inhibitor arcs of the net decides if (**FIN**) is fulfilled. Thus, the polynomial test of the TFP looks, formally, as follows:

Algorithm 6 (*Tests, Whether $\mathcal{S}$ Fulfils the TFP w.r.t. (N, m_0).*).
Step 1: Repeat for each place $p \in P$:
Step 1.1: If $(V, \prec, l)$ *fulfills the TFP w.r.t.* (P, T, F, W, m_0) *and* p do the following (let χ_p be the computed token flow function):
Repeat for each node $v \in V$:
Step 1.1.1: Compute the flow network (G, c, s, t) associated to χ_p and v.
Step 1.1.2: Compute the value $M(p, v)$ of a minimal flow in (G, c, s, t).
Step 2: Return *true* if, and only if, $(V, \prec, l)$ fulfills the TFP w.r.t. (P, T, F, W, m_0) and p for each $p \in P$ and $M(p, v) \leqslant I((p, l(v))))$ for each $p \in P$ and each $v \in V$.

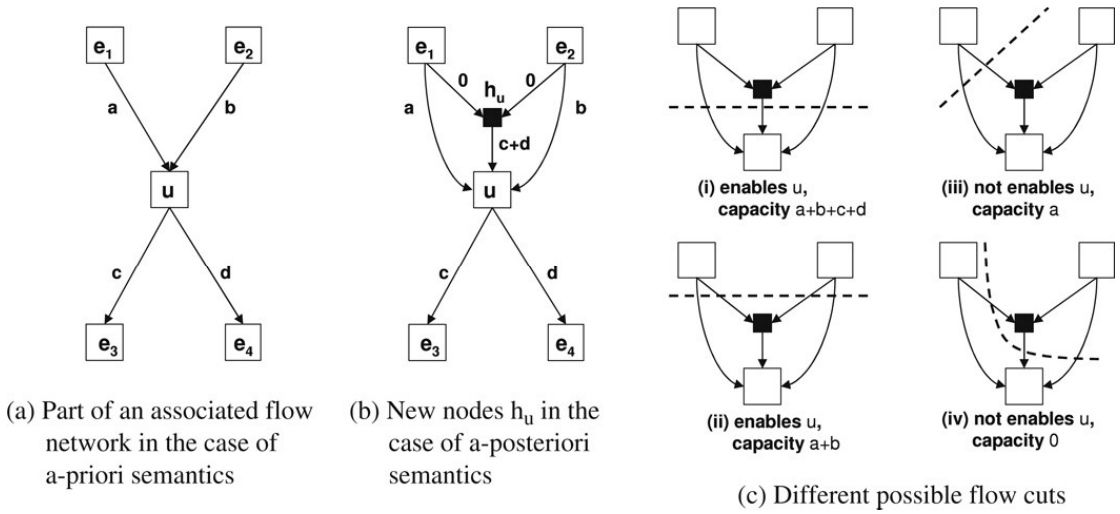
3.3. A posteriori semantics

It is easy to adapt the considerations of the last subsection to the case of a posteriori semantics. We simply use a different notion of *final marking* to reflect the more restrictive occurrence rule. For the efficient computation of inhibitor values, then, a modified version of associated flow networks is used.

3.3.1. Token flow property

If lpo is enabled w.r.t. a PTI-net in the a posteriori semantics, then for each place p there is a token flow function χ_p satisfying (**IN**) and (**OUT**). The existence of such token flow functions is not enough to ensure that lpo is enabled. This is because the execution of a prefix of lpo still might produce too many tokens in a place p (according to χ_p), disabling a subsequent transition step, which tests p via inhibitor arcs. As in the case of a priori semantics, the number of tokens in a place which is not allowed to exceed an inhibitor weight (in order not to disable transition steps subsequent to a certain prefix) is denoted as *final marking* of a prefix. It consists of the token flow on edges, leaving the prefix and the token flow produced by the subsequent cut of the prefix.

Definition 25 (*Final Marking*). Let $\mathrm{lpo}' = (V', \prec', l')$ be a prefix of $(V, \prec, l)$ and $\chi : V^0 \to \mathbb{N}$ be a token flow function of $(V, \prec, l)$.
The cut $C_{V'} = C_{\mathrm{lpo}'} = \{v \in V \setminus V' \mid (w \prec^0 v) \implies (w \in V')\}$ is called *subsequent cut of* lpo'.
The *final marking of* lpo' *(w.r.t. χ)* is defined by
$m_{\mathrm{lpo}'}(\chi) = \sum_{u \in V' \cup \{v_0\},\, v \notin V',\, u \prec^0 v} \chi((u, v)) + \sum_{v \in C_{\mathrm{lpo}'}} \mathrm{Out}(v, \chi)$.



(a) Part of an associated flow network in the case of a-priori semantics

(b) New nodes h_u in the case of a-posteriori semantics

(c) Different possible flow cuts

Fig. 11. Construction of the associated flow network in the case of a posteriori semantics from the associated flow network in the case of a priori semantics and several possible new flow cuts.

As in the case of a priori semantics, if a token flow function fulfills (**IN**) and (**OUT**) then the final marking of a prefix in fact does not depend on the concrete distribution of the token flow given by this token flow function, but only on the nodes belonging to the prefix. In this case, the final marking can be computed through $m_{\text{lpo}'}(\chi) = m_0(p) + \sum_{t\in T} l(V')(t)(W((t,p)) - W((p,t))) + \sum_{t\in T} l(C_{\text{lpo}'})(t)W((t,p))$, since $\sum_{v\in C_{\text{lpo}'}} \text{Out}(v,\chi) = \sum_{t\in T} l(C_{\text{lpo}'})(t)W((t,p))$. The notion of the TFP is now as that in the case of a priori semantics (apply Definition 17 to LPOs).

Theorem 26. lpo *is enabled w.r.t.* (N, m_0) *(a posteriori semantics) if, and only if, it fulfills the TFP w.r.t.* (N, m_0).

Proof. The proof is analogous to the proof of Theorem 19 in the case of a priori semantics. We need that Lemma 18 is also valid for LPOs, i.e. that $V' \subseteq V$ defines a prefix of lpo enabling a node $v \in V$ if, and only if, there is a step sequentialization lpo′ of lpo, such that V' defines a prefix of lpo′ enabling v. This holds since lpo can be considered as LSO $\mathcal{S} = (V, \prec, \sqsubset, l)$ with $\prec = \sqsubset$. If $\sigma = \tau_1 \dots \tau_n$ denotes the step sequence representing lpo′ as constructed in the proof of Lemma 18 and m_k denotes the marking reached after the execution of $\tau_1 \dots \tau_k$, we deduce $m_k(p) + \sum_{t\in T} \tau_{k+1}(t)W((t,p)) = m_{\text{lpo}'}(\chi_p)$, since $l(C_{V'})(t) = \tau_{k+1}(t)$. The statement now follows from:

- If lpo is enabled, then $m_k(p) + \sum_{t\in T} \tau_{k+1}(t)W((t,p)) \le I((p,t))$ for each $t \in \tau_{k+1}$.
- If lpo fulfills the TFP, then $m_{\text{lpo}'}(\chi_p) \le I((p, l(v)))$ for each $v \in C_{\text{lpo}'}$.

3.3.2. Polynomial test

The idea to derive a polynomial algorithm to test whether an LPO fulfills the TFP w.r.t. a marked PTI-net (N, m_0) in the a posteriori semantics is the same as in the case of the a priori semantics. We define an associated flow network, such that final markings of prefixes can be computed as capacities of flow cuts in the flow network. For this, we use the same notion of *inhibitor values* as before, just relating it to the modified notion of final markings. In this modified notion, the capacity of a flow cut not only need count the token flow leaving a prefix lpo′, but additionally needs to count the token flow produced by the subsequent cut $C_{\text{lpo}'}$. To count the token flow leaving lpo′, we first construct a flow network as in the case of a priori semantics (Fig. 11 (a)). To count the token flow produced by $C_{\text{lpo}'}$ we add additional nodes to this network in order to add this token flow to the capacity of cuts (Fig. 11).

Definition 27 (*Associated Flow Network*). Let $\chi : \prec^0 \to \mathbb{N}$ be a token flow function of lpo $= (V, \prec, l)$. Further, let $U_{\min}$ be the smallest prefix enabling v and $U_{\max}$ be the largest prefix enabling v.
Define the *flow network* (G, c, s, t), $G = (W, E)$, *associated with* χ *and* v, by $W = V^0 \cup H$, $s = v_0$, $t = v^\star$, $E = \prec^0 \cup F$ and $c = d \cup e$, where

- $H = \{h_u \mid u \in (U_{\max} \cup C_{U_{\max}}) \setminus U_{\min}\}$.
- $F = \{(h_u, u) \mid h_u \in H\} \cup \{(w, h_u) \mid h_u \in H,\ w \in {}^\bullet u\}$.
- $d : \prec^0 \to \mathbb{N}$ is given by $d((u,w)) = \chi((u,w))$ if $(u \in U_{\max} \cup \{v_0\} \wedge w \not\prec v)$ and $d((u,w)) = 0$ else.
- $e : F \to \mathbb{N}$ is given by $e((h_u, u)) = \text{Out}(u, \chi)$ and $e((w, h_u)) = 0$.

Let V' define a prefix of lpo. Then the flow cut $(S_{V'}, T_{V'})$ *corresponding to* V' is defined by $S_{V'} = V' \cup \{v_0\} \cup \{h_u \mid u \in C_{V'} \cup V'\}$ and $T_{V'} = W \setminus S_{V'}$.

Lemma 28. *Let* lpo′ $= (V', \prec', l')$ *be a prefix enabling a node* v. *Further, let* χ *be a token flow function of* lpo *and* (G, c, s, t), $G = (W, E)$, *be the flow network associated to* χ *and* v. *Then* $m_{\text{lpo}'}(\chi) = c((S_{V'}, T_{V'}))$.

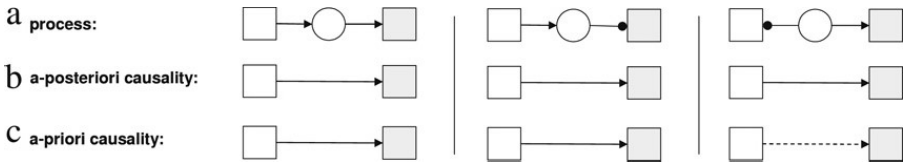


Fig. 12. Causality relations between events corresponding to the relation between these events in process nets.

Proof. Easy computation using $(x, y) \in E \cap (S_{V'} \times T_{V'}) \Leftrightarrow (x \prec^0 y \wedge x \in V' \cup \{v_0\} \wedge y \in V \setminus V') \vee (x = h_u \wedge y = u \wedge u \in C_{V'}) \vee (x \in {}^\bullet v \cap (V' \cup \{v_0\}) \wedge y = h_u \wedge u \notin (V' \cup C_{V'}))$ (Fig. 11 (c)).

Theorem 29. *Let v be a node, and χ be a token flow function of* lpo $= (V, \prec, l)$. *Further, let (G, c, s, t), $G = (W, E)$, be the flow network associated to χ and v. Then $Inh(v, \chi) = \max\{c((S, T)) \mid (S, T)$ flow cut of $(G, c, s, t)\}$.*

Proof. The proof is analogous to the proof of Theorem 24 in the case of a priori semantics. The idea is to show that, if (S, T) is a flow cut of (G, c, s, t) not corresponding to a prefix enabling v, then there is a flow cut corresponding to a prefix enabling v with bigger capacity. Then the statement follows from Lemma 28. In comparison to the proof of Theorem 24 we now must account for flow cuts separating the h_v-nodes from other nodes, in different ways.

For a flow cut (S, T) of (G, c, s, t) we set $V' = S \cap V^0$, $S' = S \cap V^0$ and $T' = V^0 \setminus S$. Then, by construction, (S', T') is a flow cut of the associated flow network in the case of a priori semantics. We can distinguish the following cases.

If V' does not define a prefix of lpo, then analogous to the case of a priori semantics, it follows that there are nodes $u \in S' \subseteq S$ and $w \in T' \subseteq T$ with $(w, u) \in \prec^0 \subseteq E$, i.e. $c((S, T)) = 0$.

Let lpo$' = (V', \prec, l)$ be a prefix of lpo. First consider the case $S \neq S_{V'}$ (for the definition of $S_{V'}$ see Definition 27). That means, one of the following statements holds:

(i) $\exists v' \in V' : h_{v'} \notin S$: In this case we deduce $(h_{v'}, v') \in (T \times S) \cap E$, i.e $c((S, T)) = 0$.
(ii) $\exists v' \in C_{V'} : h_{v'} \notin S$: In this case we deduce $c((S \cup \{h_{v'}\}, T \setminus \{h_{v'}\})) = c((S, T)) + c((h_{v'}, v')) \geqslant c((S, T))$ (Fig. 11(c)(i),(c)(ii)).
(iii) $\exists v' \in T' \setminus C_{V'} : h_{v'} \in S$: In this case we deduce $(w, h_{v'}) \in (T \times S) \cap E$ for some $w \in {}^\bullet v'$, i.e $c((S, T)) = 0$ (Fig. 11(c)(iv)).

Now assume that lpo$' = (V', \prec, l)$ is a prefix of lpo with $S = S_{V'}$. If lpo$'$ does not enable v, then analogously to the case of a priori semantics, it follows that there is a prefix enabling v whose corresponding flow cut has a bigger capacity than (S, T) (Fig. 11(c)(iii)). If lpo$'$ enables v, then (S, T) corresponds to a prefix enabling v.

The algorithm looks like that in the last paragraph, just relating to the different notion of associated flow network.

4. Other net classes

In this section, we briefly discuss how to adapt the presented theory to the net classes of elementary nets, elementary nets with (mixed) context (in the a posteriori and a priori semantics), p/t-nets with capacities (in the weak and strong semantics) and p/t-nets with unweighted inhibitor arcs (in the a posteriori and a priori semantics).

The construction for elementary nets (as mentioned in the Introduction) can also be applied to *elementary nets with (mixed) context*, that means to elementary nets extended by read arcs and/or (unweighted) inhibitor arcs. Processes of such nets are defined w.r.t. their so called complementation (adding a complement place for each place in order to get a contact free net). Processes additionally contain read arcs between events and conditions to reflect the read and inhibitor arcs of the net. To represent inhibitor arcs, read arcs connected to complement places are used. Read arcs in a process, directly refer to read and inhibitor arcs in the net. The run corresponding to a process is given by an LPO in the case of a posteriori semantics, and by an LSO in the case of a priori semantics. Given an LPO lpo (LSO $\mathcal{S}$), we try to construct a process whose corresponding run is sequentialized by lpo ($\mathcal{S}$) in the same way as above. Again, there is always, at most, one possibility to append an event. If it is not possible to append the event, or if appending the event produces order not existent in lpo ($\mathcal{S}$), lpo ($\mathcal{S}$) is no execution. The only difference is that now several possibilities to generate order between events have to be checked, because not only token flow generates order ("earlier than"), but also context relations generate order ("earlier than" or "not later than", depending on the considered semantics, see Fig. 12). Obviously, this construction again needs linear time.

Clearly, the theory presented in this paper can be applied to *p/t-nets with unweighted inhibitor arcs* (that means having the weight 0), since they are a special case of PTI-nets. We simply have to check if the inhibitor value of events exceeds the value 0 w.r.t. inhibitor places. It is not necessary to apply a flow minimization algorithm here, because the maximal capacity of a flow cut in the associated flow network is 0 if, and only if, all capacities are 0. Therefore, it is enough to construct the associated flow network and to check the capacity function.

Finally, let us consider *p/t-nets with capacities*, i.e. p/t-nets where each place p has an upper (capacity) bound $K(p) \in \mathbb{N}$ for the number of tokens which it can carry. There are several semantics of such nets.

- According to *weak semantics* [34] (resp. capacities of type E2 given in [35]), given a marking m enabling a transition t, t first consumes the tokens given by $W((p, t))$ yielding an intermediate marking $m(p) - W((p, t))$ in places p and then produces the tokens given by $W((t, p))$ yielding the marking $m(p) - W((p, t)) + W((t, p))$. There are two concurrent step

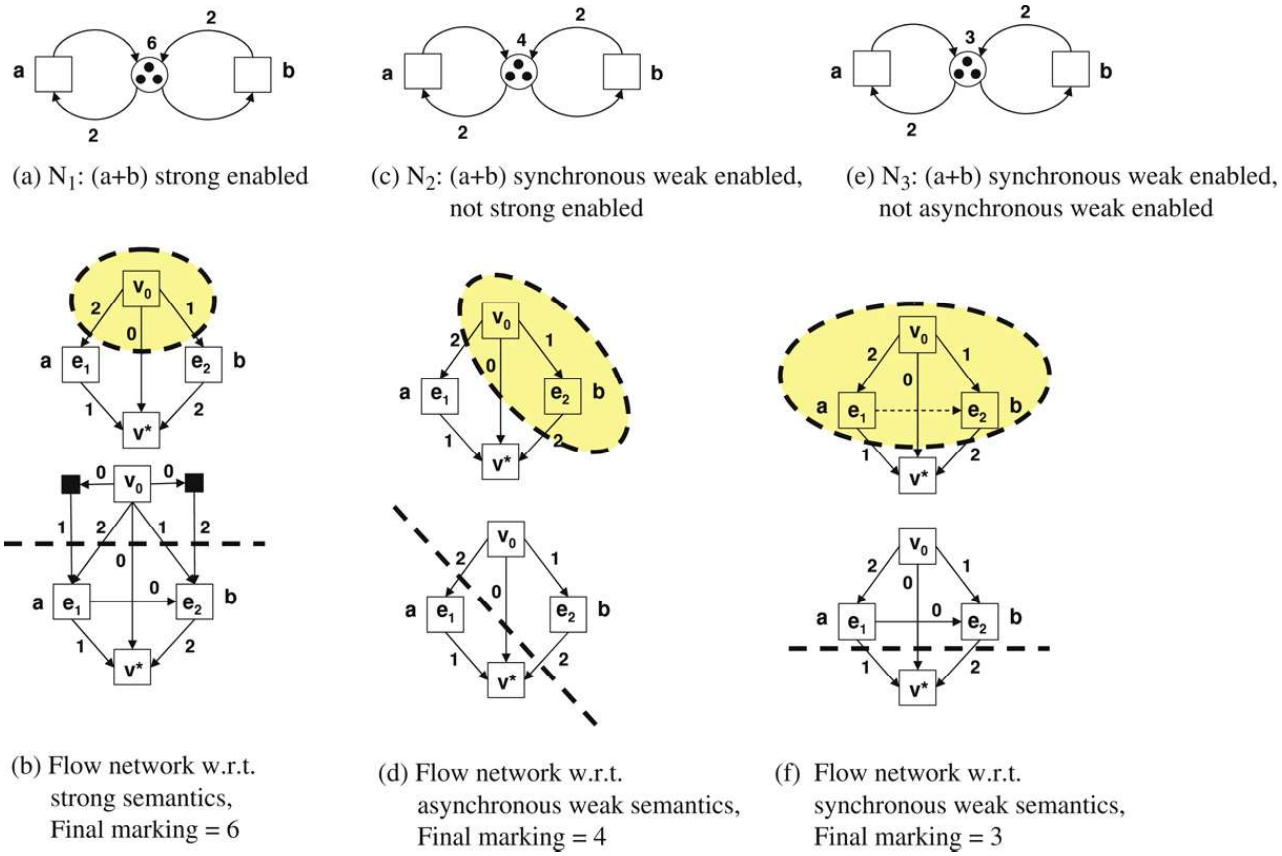


Fig. 13. Enabledness of the step $(a + b)$ w.r.t. different semantics and p/t-nets with varying capacities. The step $(a + b)$ is represented in case (a) and (c) by an LPO ((b) and (d) upper part), in case (c) by an LSO ((f) upper part). In each case, a token flow and a prefix defining a maximal final marking is shown. Finally, the associated flow network is illustrated, together with the flow cut corresponding to the prefix ((b), (d), (f) lower part).

semantics for weak capacities to distinguish, namely *asynchronous concurrent step semantics* and *synchronous concurrent step semantics*.

- A multi-set (a step) of transitions τ is *asynchronous enabled to occur in a marking* m if $m(p) \geq \sum_{t\in T} \tau(t)W((p,t))$ and $K(p) \geq m(p) - \sum_{t\in T} \tau'(t)(W((p,t)) - W((t,p)))$ for each place p and each multi-set of transitions τ' with $\forall t \in T : \tau'(t) \leqslant \tau(t)$. This ensures that if a step is enabled to occur, all sub-steps are also enabled to occur. In other words, the transition occurrences in such a step are concurrent (Fig. 13(c)).
- A multi-set (a step) of transitions τ is *synchronous enabled to occur in a marking* m if $m(p) \geq \sum_{t\in T} \tau(t)W((p,t))$ and $K(p) \geq m(p) - \sum_{t\in T} \tau(t)(W((p,t)) - W((t,p)))$ for each place p. It is *not* required that if a step is enabled to occur, all sub-steps are also enabled to occur. The transitions in such a step need not be concurrent and it is possible to distinguish concurrent and synchronous behavior (Fig. 13(e)).

- According to the *strong semantics* [34] (resp. capacities of type E1 given in [35]), given a marking m enabling a transition t, t can consume and produce tokens in any order, i.e. it behaves either as in the case of weak semantics or it first produces tokens given by $W((t,p))$ yielding an intermediate marking $m(p)+W((t,p))$ and then consumes tokens given by $W((p,t))$ yielding the marking $m(p)-W((p,t))+W((t,p))$. The concurrent step semantics in this case is defined as follows. A multi-set (a step) of transitions τ is *strong enabled to occur in a marking* m if $m(p) \geq \sum_{t\in T} \tau(t)W((p,t))$ and $K(p) \geq m(p) + \sum_{t\in T} \tau(t)W((t,p))$ for each place p (Fig. 13(a)).

In [35] it is shown that given a p/t net with capacities with an initial marking m_0, for the strong semantics and for the asynchronous weak semantics there exists a transformation into a marked p/t net with the same number of transitions, such that the step sequences of the net with capacities and the transformed net without capacities are equal.[6] The processes and runs of the transformed net provide, then, the non-sequential semantics of the p/t-net with capacities. We deduce that in both cases, causal semantics can be given equivalently as enabled LPOs or as executable LPOs (as defined for p/t-nets). To test whether a given LPO is an execution of such a p/t-net with capacities, we can apply the verification algorithm developed for p/t-nets to the transformed net.

But it is also possible to characterize enabled LPOs in these cases (strong and asynchronous weak semantics) directly through an adapted TFP w.r.t. the original net. Clearly this adapted TFP again includes the TFP for p/t-nets. Additionally, we

6 For strong capacities, the transformation is analogous to the complementation of elementary nets, while for weak capacities the transformation is more complicated.

have to account for the capacity constraints. These are very similar to the inhibitor constraints in the case of PTI-nets. We just replace inhibitor bounds by capacity bounds and require that some appropriately defined *final marking* of a prefix should not exceed the capacity bound. Finally, we construct a flow network, such that final markings correspond to capacities of flow cuts in this network.

For the asynchronous weak semantics, the definition of final markings and the associated flow network is very similar as for PTI-nets w.r.t. the a priori semantics. Observe that the capacity constraint of the step occurrence rule can be translated into the requirement that the number of tokens in a place after the occurrence of an arbitrary prefix (according to some token flow function fulfilling (**IN**) and (**OUT**) w.r.t. some place) should not exceed the capacity bound of the considered place. Note here, that each step of transition occurrences corresponds to such a prefix and vice versa in the sense that the prefix enables the step in the LPO (this way sub-steps are included in a natural way). That means the *final marking* of a prefix can be defined as for PTI-nets w.r.t. the a priori semantics (remember that LPOs are special LSOs). The only difference is that we do not consider the inhibitor values of all events of the LPO lpo, but only the inhibitor value of lpo as a whole. This *inhibitor value* is defined as the maximum over all final markings of prefixes of lpo. It can be computed as the maximal capacity of a flow cut in an associated flow network. This flow network is defined analogously as for PTI-nets w.r.t. the a priori semantics (applied to LPOs), with the only difference being that the capacity of *all* edges is computed from the token flow function and no capacity is explicitly set to 0 (Fig. 13(d)).

For the strong semantics, the definition of final markings and the associated flow network is very similar as for PTI-nets w.r.t. the a posteriori semantics. Observe that the capacity constraint of the step occurrence rule can be translated into the requirement that the number of tokens in a place after the occurrence of an arbitrary prefix lpo' of the given LPO increased by the number of tokens produced by the subsequent step $C_{\mathrm{lpo}'}$, should not exceed the capacity bound of the considered place. That means the *final marking* of a prefix can be defined as for PTI-nets w.r.t. the a posteriori semantics. As for the asynchronous weak semantics, we continue by considering the inhibitor value of lpo as a whole, defined as the maximum over all final markings of prefixes of lpo. It can be computed as the maximal capacity of a flow cut in an associated flow network. This flow network is defined analogously as for PTI-nets w.r.t. the a posteriori semantics (applied to LPOs) with the only difference being that for *all* nodes v, a node h_v is added, and the capacity of *all* edges not relating to a node h_v is computed as a sum of token flows (Fig. 13 (b)).

The synchronous weak semantics executions are given by LSOs, since here concurrent and synchronous occurrence of transitions can be distinguished. Enabled LSOs are defined in the same way as for PTI-nets using (synchronous) step occurrence sequences. Enabled LSOs can be equivalently characterized through an adapted token flow property which can be defined analogously as in the case of asynchronous weak semantics. That means the definition of final markings and the associated flow network is the same as for asynchronous weak semantics, just applied to general LSOs (Fig. 13 (f)).

5. Conclusion

In this paper, we have presented several *polynomial* algorithms to verify *partially ordered executions* of Petri nets for several Petri net classes, including p/t-nets, p/t-nets with inhibitor arcs and p/t-nets with capacities. These algorithms are based on the new formal concept of *token flow functions* to represent non-sequential semantics of Petri nets. For p/t-nets we implemented the presented algorithms into the tool VIPtool.

The presented verification concept cannot be compared directly to verification concepts presented so far in literature. Whereas we ask whether a given scenario represents valid behavior of a given net, usually there are constructed behavior models of a given net (such as the reachability graph or the unfolding), which are then verified to fulfill certain requirements (given for example by temporal formulas) [36,37].

The paper summarizes, consolidates and extends two conference papers. In [1], for the first time the new concept of *token flow functions* was presented for p/t-nets, leading to polynomial algorithms to verify executions and minimal executions of p/t-nets. The content of [1] is presented in this paper in a consolidated manner. It is extended by a second more efficient algorithm, a discussion of possible optimizations, a comparison of the algorithms concerning time complexity and fault analysis and a discussion of strict executions. In [2] we extended the theory to inhibitor nets, considering their a priori semantics. These considerations are extended in this paper by also examining their a posteriori semantics and different semantics of p/t-nets with capacities.

Acknowledgments

This paper was supported by the project SYNOPS of the German Research Council.

References

[1] G. Juhás, R. Lorenz, J. Desel, Can I execute my scenario in your net?, in: G. Ciardo, P. Darondeau (Eds.), ICATPN, in: Lecture Notes in Computer Science, Springer, 2005, pp. 289–308.
[2] R. Lorenz, S. Mauser, R. Bergenthum, Testing the Executability of Scenarios in General Inhibitor Nets, in: ACSD, IEEE Computer Society, 2007, pp. 167–176.
[3] R. Lorenz, Szenario-basierte Verifikation und Synthese von Perinetzen: Theorie und Anwendungen, Habilitation, 2006.

[4] J. Billington, M. Diaz, G. Rozenberg (Eds.), Application of petri nets to communication networks, advances in petri nets, in: Application of Petri Nets to Communication Networks, in: Lecture Notes in Computer Science, vol. 1605, Springer, 1999.
[5] B.-H. Schlingloff, A. Martens, K. Schmidt, Modeling and model checking web services, Electronic Notes Theoretical Computer Science 126 (2005) 3–26.
[6] M.C. Zhou, F. Di Cesare, Petri Net Synthesis for Discrete Event Control of Manufacturing Systems, Kluwer, 1993.
[7] W.M.P. van der Aalst, K. van Hee, Workflow Management: Models, Methods, and Systems, MIT Press, Cambridge, Massachsetts, 2002.
[8] J.L. Peterson, Petri Net Theory and the Modeling of Systems, Prentice-Hall, 1981.
[9] S. Donatelli, G. Franceschinis, Modelling and Analysis of Distributed Software using GSPNs, in: Petri Nets (2), in: LNCS, 1998, pp. 438–476.
[10] U. Goltz, W. Reisig, The non-sequential behaviour of petri nets, Information and Control 57 (2/3) (1983) 125–147.
[11] Ugo Montanari, Francesca Rossi, Contextual nets, Acta Informatica 32 (6) (1995) 545–596.
[12] N. Busi, G. M. Pinna, Process semantics for place/transition nets with inhibitor and read arcs, Fundamental Informaticae 40 (2-3) (1999) 165–197.
[13] H.C.M. Kleijn, M. Koutny, Process semantics of general inhibitor nets, Information and Computation 190 (1) (2004) 18–69.
[14] G. Juhás, R. Lorenz, S. Mauser, Complete process semantics for inhibitor nets, in: J. Kleijn, A. Yakovlev (Eds.), ICATPN, in: Lecture Notes in Computer Science, Springer, 2007, pp. 184–203.
[15] V. Pratt, Modelling concurrency with partial orders, International Journal of Parallel Programming 15 (1986) 33–71.
[16] J. Grabowski, On partial languages, Fundamenta Informaticae 4 (2) (1981) 428–498.
[17] A. Kiehn, On the interrelation between synchronized and non-synchronized behaviour of petri nets, Elektronische Informationsverarbeitung und Kybernetik 24 (1/2) (1988) 3–18.
[18] W. Vogler, Modular Construction and Partial Order Semantics of Petri Nets, in: Lecture Notes in Computer Science, Springer, 1992.
[19] A. Goldberg, S. Rao, Beyond the flow decomposition barrier, Journal of the ACM 45 (5) (1998) 783–797.
[20] R. Janicki, M. Koutny, Semantics of inhibitor nets, Information and Computation 123 (1) (1995) 1–16.
[21] H. C. M. Kleijn, M. Koutny, Process semantics of general inhibitor nets, Information and Computation 190 (1) (2004) 18–69.
[22] H. Gaifman, V.R. Pratt, Partial Order Models of Concurrency and the Computation of Functions, in: LICS, IEEE Computer Society, 1987, pp. 72–85.
[23] R. Janicki, M. Koutny, Invariants and paradigms of concurrency theory, in: E.H.L. Aarts, J. v. Leeuwen, M. Rem (Eds.), PARLE (2), in: Lecture Notes in Computer Science, Springer, 1991, pp. 59–74.
[24] D. Jungnickel, Graphs, Networks and Algorithms, 2nd edition, in: Algorithms and in Mathematics, Springer, 2000.
[25] V.M. Malhotra, M.P. Kumar, S.N. Maheshwari, An $o(|v|^3)$ algorithm for finding maximum flows in networks, Information Processing Letters 7 (6) (1978) 277–278.
[26] A.V. Goldberg, R.E. Tarjan, A new approach to the maximum-flow problem, Journal of the ACM 35 (4) (1988) 921–940.
[27] R.K. Ahuja, J.B. Orlin, R.E. Tarjan, Improved time bounds for the maximum flow problem, SIAM Journal on Computing 18 (5) (1989) 939–954.
[28] L.R. Ford, D.R. Fulkerson, Maximal flow through a network, Canadian Journal of Mathematics 8 (1956) 399–404.
[29] R.K. Ahuja, A.V. Goldberg, J.B. Orlin, R.E. Tarjan, Finding minimum-cost flows by double scaling, Mathematical Programming 53 (1992) 243–266.
[30] T. Watanabe, Y. Mizobata, K. Onaga, Legal Firing Sequence and Related Problems of Petri Nets., in: PNPM, IEEE Computer Society, 1989, pp. 277–286.
[31] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows — Theory, Algorithms, and Applications, Prentice-Hall, 1993.
[32] N. Busi, G.M. Pinna, Non sequential semantics for contextual p/t nets, in: Jonathan Billington, Wolfgang Reisig (Eds.), Application and Theory of Petri Nets, in: Lecture Notes in Computer Science, Springer, 1996, pp. 113–132.
[33] N. Busi, G.M. Pinna, Comparing truly concurrent semantics of contextual place/transition nets, Fundamental Informaticae 44 (2) (2000) 209–244.
[34] J. Desel,W.Reisig, Place/Transition Petri Nets, in: Reisig, W. Rozenberg (Eds.), Petri Nets, in: Lecture Notes in Computer Science, Springer, 1998, pp. 123–174.
[35] R. Devillers, The semantics of capacities in P/T nets, in: Advances in Petri Nets 1989, in: LNCS, 1989, pp. 128–150.
[36] J. Esparza, K. Heljanko, Implementing LTL model checking with net unfoldings., in: M.B. Dwyer (Ed.), SPIN, in: Lecture Notes in Computer Science, Springer, 2001, pp. 37–56.
[37] C. Schröter, S. Schwoon, J. Esparza, The model-checking kit., in: W.M.P. van der Aalst, E. Best (Eds.), ICATPN, in: Lecture Notes in Computer Science, Springer, 2003, pp. 463–472.